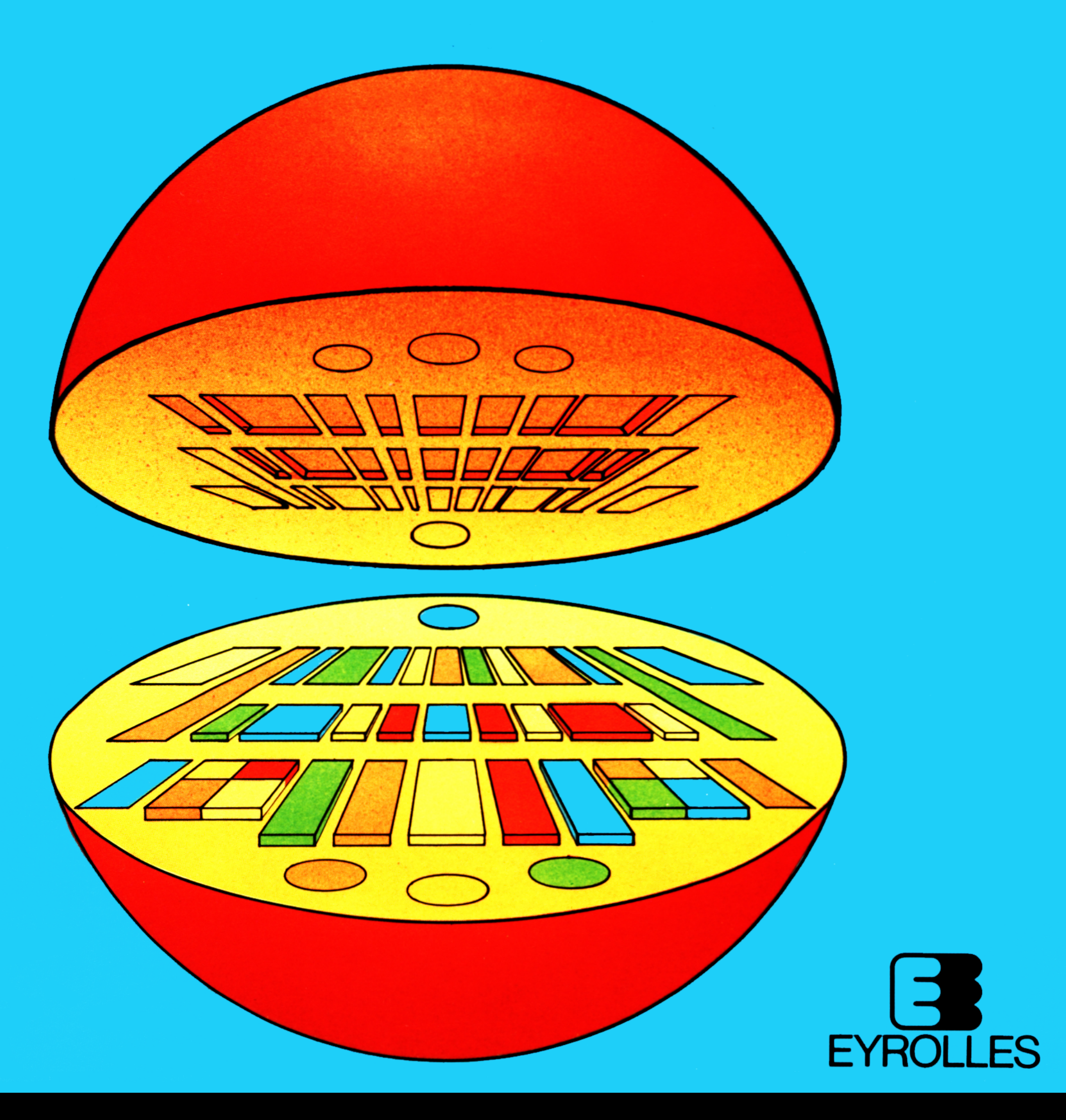
TOUT SAVOIR SUR

LYNX



EYROLLES

BRUNO VANRYB. ROGER POLITIS

TOUT SAVOIR SUR

LYNX

CHEZ LE MÊME ÉDITEUR

Dans la même collection

VANRYB-POLITIS	—	*Tout savoir sur New-Brain*, 104 p.; 1984.
GOLDSTEIN	—	*Le guide de l'IBM-PC*, 272 p.; 1984.
BUNN	—	*Obtenez le maximum de votre ATARI*, 104 p., 1984.
GARCIA	—	*Le Basic minimum.* 15 mots pour apprendre à programmer, 104 p., 1984.
WANNER	—	*Aller plus loin en BASIC TO7*, 312 p., 1984.
POLITIS-VANRYB	—	*Tout savoir sur ORIC*, 168 p., 1984.
VANRYB-POLITIS	—	*Tout savoir sur Multitech*, 152 p., 1984.
DELANNOY	—	*Faites vos jeux avec ORIC*, 224 p., 1984.
IFRAH	—	*Faites vos jeux avec CANON XO7*, 104 p., 1984.
RAVEL	—	*Tout savoir sur LASER 200-210*, 112 p., 1984.
VANRYB-POLITIS	—	*Tout savoir sur LYNX*, 176 p., 1984.
POLITIS-VANRYB	—	*Tout savoir sur ATMOS*, 192 p., 1984.

COLLECTION MICROPLUS

TOUT SAVOIR SUR LYNX

par

Bruno VANRYB
Roger POLITIS



61, boulevard Saint-Germain — 75005 PARIS
1984

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service « Presse », Éditions EYROLLES
61, Boulevard Saint-Germain,
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

Introduction

Le Lynx est l'un des premiers ordinateurs dits "familiaux" à présenter en version de base un Basic réellement complet et comparable à celui de plus gros systèmes, associé à une exploitation aisée et performante de la couleur et du graphique.

Ce Basic particulier et la présence d'instructions originales parfois issues d'autres langages comme le Pascal (PROC, DEFPROC), font de cette machine un outil réellement puissant et polyvalent.

Présenté sous la forme d'un manuel de Basic approfondi, ce livre est destiné à ceux qui, déjà au fait des rudiments du Basic, désirent aller plus loin dans la connaissance de leur ordinateur. De nombreux programmes, dont certain sont des logiciels complets et prêts à utiliser, permettent d'ouvrir la voie vers une programmation avancée.

Citons un générateur de caractères incluant la sauvegarde sur cassette des jeux créés, ainsi qu'un programme musical prévu pour mémoriser une mélodie, en éditer la partition et copier celle-ci sur imprimante ou cassette.

L'initiation au langage-machine n'a pas été oubliée: Une présentation simple et quelques "astuces" utiles permettront au lecteur, même novice en la matière, d'exploiter pleinement les accès directs à la mémoire en modes texte ou graphique. On trouvera, parmi les programmes proposés, une méthode de reconfiguration du clavier, ainsi que la routine à utiliser pour créer et exploiter deux pages-écran.

Table des matières

1

Le Basic Lynx

1.1. *Introduction*

Bien que toutes les commandes du Basic du Lynx soient décrites dans ce chapitre, seules seront discutées en détail celles dont la syntaxe ou l'utilisation présente des particularités, la référence étant le Basic Microsoft 5.Ø, le plus répandu actuellement. Des petits programmes très simples viendront s'ajouter à ces explications dans le cas de commandes nécessitant une présentation plus complète.

D'autre part, et malgré un Basic particulièrement étendu, certaines commandes sont absentes du Lynx. Cette section se termine par quelques sous-programmes palliant à ces absences lorsque cela est possible.

REMARQUE : Les instructions concernant le langage-machine, la couleur, le graphique et le son seront traitées dans les chapitres qui leur sont consacrés (§ 2.4.2, 3.1.3 et 4.1 respectivement).

1.2. Conventions d'écriture

Pour faciliter la compréhension, nous présenterons les instructions et fonctions de la manière suivante :

Syntaxe : Donne la syntaxe complète de l'instruction ou fonction.

Application(s) : Donne le détail de l'utilisation de la commande.

REMARQUES : Dans cette rubrique seront exposées les particularités de la commande, ou les différences avec le Basic Microsoft.

Exemple(s) : Selon la complexité des cas, un ou plusieurs exemples seront donnés.

— Tous les mots écrits en majuscules représentent les mots-clés du Basic, et sont à entrer tels quels au clavier.

— Les mots écrits en minuscules et compris entre < > sont les données à fournir par le programmeur.

— Les données facultatives sont incluses entre crochets [ ].

— Toutes les ponctuations, hormis les deux précédentes, font partie intégrante de la syntaxe et doivent être tapées telles quelles.

— En ce qui concerne les nombres, nous adopterons les conventions suivantes :

- X, Y, Z représentent des nombres en virgule flottante,
- N, M des nombres entiers,
- A représente une adresse mémoire (entier compris entre Ø et 65535).

Exemple : INPUT [<"commentaire";>] <variable 1> [<,variable 2>]

— L'instruction INPUT est la commande Basic.

— Le commentaire est optionnel, sinon il doit être entre guillemets " " et suivi de ; .

— Au moins une variable doit être donnée comme argument.

— On peut si on le désire mentionner plusieurs variables, en les séparant par des virgules.

— Il est correcte d'écrire :
INPUT "Donnez votre âge et votre mois de naissance" ;A,M$.

1.3. *Quelques rappels*

Avant toutes choses, nous pensons qu'il est utile de préciser quelques notions de base, sur le fonctionnement des micro-ordinateurs, et celui du Lynx en particulier.

Le Lynx utilise un langage évolué, le Basic, pour communiquer avec nous. Les ordres donnés en Basic sont ensuite interprétés et exécutés un à un. Le programme interne, chargé de cette traduction s'appelle l'**interpréteur Basic**, et il a été écrit une fois pour toutes : On ne peut pas le modifier, et la mémoire le contenant s'appelle **mémoire morte** ou **MEM** en français, et **ROM** (Read Only Memory) en anglais. Le langage Basic comprend des **Instructions**, des **Fonctions**, des **Opérateurs** et des **Opérandes** ou **Arguments**. Expliquons ces termes

1.3.1. Instructions et fonctions

— Une **Instruction** est un mot-clé compris par le Basic, qui implique l'exécution d'une tâche précise par l'ordinateur. Certaines instructions doivent être suivies d'un argument, certaines non.

Exemples : Dans PRINT "BONJOUR", PRINT est l'inscription et "BONJOUR" est l'argument. Mais PRINT peut aussi être utilisé seul.

Par contre, CLS (effacement de l'écran) est une instruction utilisée toujours seule, alors que INPUT implique nécessairement l'emploi d'un ou plusieurs arguments.

— Une **Fonction** est également un mot-clé, mais il s'agit plutôt d'une "question" que l'on pose à l'ordinateur : Celui-ci doit nous retourner une valeur en réponse. Une fonction s'utilise toujours à la suite d'une instruction, en général PRINT ou LET.

Exemples: Dans PRINT SIN(PI/2), SIN est la fonction et PI/2 son argument.

LET A=INT(1ØØ.25) donnera à la variable A la valeur entière de l'argument, ici le nombre 1ØØ.25.

1.3.2. Variables et constantes

— Les **Opérandes** sont les valeurs, nombres ou séquences de caractères, servant d'argument aux instructions et fonctions. Les opérandes numériques sont de deux sortes :

Les constantes, qui peuvent s'écrire en notation ordinaire (1ØØ, 3.1416, etc.) ou scientifique (2.2E+3, E.7E−12).

Les variables, qui contrairement aux constantes peuvent changer de valeur en cours de programme. Leur valeur peut être définie par l'utilisateur, ou représenter le résultat d'un calcul. Une variable numérique est définie par un nom comportant au maximum un caractère alphanumérique.

Leur valeur peut être comprise entre 9.9E63 et 9.9E−63. A, N, C sont des noms de variables numériques.

Nous avons enfin les opérandes littérales, qui ne sont plus considérées comme des nombres, mais comme des suites de caractères. Il s'agit bien sûr des **chaînes de caractères**, et elles aussi sont divisées en constantes et variables.

Une *constante de chaîne* est une suite quelconque de caractères (jusqu'à 127), toujours incluse entre guillemets " " pour distinguer d'un nom de variable. "PROGRAMME", "TOTO", "123456789" sont des constantes chaînes.

Les noms de *variables de chaîne* suivent les mêmes règles, mais doivent être suivis du symbole $. A$, F$ sont des variables chaînes.

Remarquons que tous les types de variables peuvent servir à constituer des tableaux. Par exemple, A(3*3) représente un tableau de nombres à deux dimensions que l'on peut assimiler à un damier de 3×3=9 cases, chacune d'entre elles contenant un nombre. La lecture et l'écriture dans ces cases peut ensuite s'effectuer de manière directe en appelant un élément du tableau par son numéro, ou de manière séquentielle en lisant le tableau du premier au dernier élément.

Les nombres entre parenthèses, permettant d'appeler individuellement chaque élément d'un tableau s'appellent des **indices** ; un élément de tableau s'appelle une variable indicée, et il est à noter que les indices peuvent être des constantes ou des variables. Le Lynx accepte les tableaux comportant un nombre quelconque de dimensions. Bien entendu, tous les éléments d'un tableau doivent être de même nature : Variables numériques ou chaînes de caractères. Voir aussi l'instruction DIM, paragraphe 1.4.2.

1.3.3. Les opérateurs

Les **Opérateurs** sont de trois sortes :

Les opérateurs arithmétiques, que tout le monde connaît bien : +, —, * (multiplié par), / (divisé par), et ** (élévation à une puissance). Ces opérateurs s'utilisent de manière normale sur constantes et variables numériques, mais remarquons que "+" peut aussi s'utiliser sur les chaînes. Les chaînes ne sont pas additionnées, mais "concaténées", c'est-à-dire mises l'une à la suite de l'autre.

Exemple : Si A=123 et B=456, alors A+B=579. Par contre, si A$="123" et B$="456", A$+B$="123456".

Les opérateurs relationnels, >, <, =, >=, <=, < > (supérieur, inférieur, égal, supérieur ou égal, inférieur ou égal, différent). Ces opérateurs peuvent s'utiliser sur des nombres. Sur le Lynx, seuls deux d'entre eux fonctionnent sur les chaînes, > et =. Dans le cas de deux chaînes, celles-ci sont comparées caractère par caractère, et sont classées par ordre alphabétique comme dans un dictionnaire. Une condition VRAIE vaut 1, et la condition FAUSSE Ø.

Exemples :

5<2 est faux, si A=(5<2), donc A=Ø.

"TOTO">"COCO" est vrai, donc l'expression (5<2)=("COCO">"TOTO") est vraie.

Les opérateurs logiques, AND, OR et NOT (ET, OU et NON) s'utilisent pour combiner entre elles des expressions utilisant des opérateurs relationnels.

NOT combiné avec = permet de simuler <> dans le cas de comparaisons de chaînes de caractères; NOT A$=B$ est équivalent à A$< >B$. Enfin, BNAND, BNOR et BNXOR effectuent une comparaison Bit à Bit de deux nombres. (BNXOR est un OR exclusif.)

Exemple :

IF A>2 AND A<5 THEN PRINT A équivaut à : SI A EST COMPRIS ENTRE 2 ET 5 ALORS PRINT A.

1.3.4. Mode direct et mode programme

Dans tout ce qui va suivre, nous nous référerons constamment au mode "direct" et au mode "programme". Rappellons que toute instruction ou séquence d'instructions entrée au clavier et non précédée d'un nombre, est exécutée dès que la touche RETURN est tapée : C'est le mode direct.

Par contre, si une séquence d'instructions est précédée d'un nombre, celui-ci sera considéré comme un numéro de ligne. Frapper RETURN ne déclenchera pas le déroulement du programme, mais mémorisera les instructions pour une exécution ultérieure : C'est le mode programme ; il faudra taper RUN pour lancer l'exécution.

Une facilité particulière est prévue en mode direct : Les touches alphanumériques du clavier génèrent les mots-clés les plus usuels du Basic, en les combinant avec ESC. Cette facilité accélère la frappe des programmes, et s'appelle le One-Key Basic. (Voir § 2.6.2.)

1.3.5. Les limites du Lynx

En dehors de ces quelques rappels, valables pour la grande majorité des Basic, les ordinateurs ont des particularités de syntaxe qu'il faut connaître pour pouvoir écrire un programme même très simple, et le Lynx ne fait pas exception à cette règle :

— Une ligne de programme comprend au maximum 24Ø caractères.

— On ne peut écrire qu'une seule instruction par ligne, mais une instruction donnée pourra contenir plusieurs paramètres :

```
1Ø LET A=1:PRINT A:LET B=2:PRINT B
```
provoque un message d'erreur.

```
1Ø LET A=1,B=2
2Ø PRINT A,B
```
sont des syntaxes correctes.

— La tentative d'imprimer ou d'utiliser une variable non préalablement définie provoque une erreur. Toute variable numérique doit donc être initialisée à Ø et toute variable chaîne à " ":

```
1Ø LET A=Ø,B$+""
2Ø PRINT A,B$
```
est correct.

— Les noms de variables ne peuvent comporter qu'un seul caractère, mais les lettres majuscules et minuscules sont reconnues comme différentes. De plus, une variable chaîne peut avoir le même nom qu'une variable numérique ou qu'un tableau sans qu'il y ait confusion entre elles. A, a, A$, a$, A(1), a(1), A$(1) (1) sont considérées comme des variables différentes.

— Les chaînes de caractères ont une longueur maximale de 127 caractères, et toute chaîne dont la longueur dépasse 16 caractères doit être dimensionnée au préalable.

— La structure des tableaux autorise un nombre quelconque de dimensions pour les tableaux numériques, et une seule pour les tableaux de chaînes.

— Une fois définie, une variable ne peut être supprimée de la mémoire sans lui affecter une nouvelle valeur. Seule, RUN réinitialise toutes les variables.

— Les numéros de ligne peuvent prendre des valeurs décimales:

```
1ØØ REM
1ØØ.5 REM
1Ø4 REM
```
sont des numéros de lignes valides.

1.4. *Les instructions du Basic*

Leur classement est effectué par ordre alphabétique, à l'intérieur de quatre grandes rubriques, correspondant aux principaux types de commandes du Basic Microsoft :

1.4.1 Commandes système	*1.4.2 Usage général*	*1.4.3 Branchements*	*1.4.4 Périphériques*
AUTO	CCHAR	FOR/NEXT/STEP	APPEND
CLS	CLR	GOSUB-RETURN	EXT SPRINT
CONT	DATA	GOTO	EXT PPRINT
DEL	DIM	LABEL	LINK ON
END	ERROR	PROC/DEFPROC	LINK OFF
LIST	IF/THEN/ELSE	REPEAT/UNTIL	LLIST
NEW	INPUT	WHILE/WEND	LOAD
RENUM	LET		LPRINT
RUN	PAUSE		MLOAD
SPEED	PRINT		SAVE
TRACE ON	RANDOM		TAPE
TRACE OFF	READ		VERIFY
	REM		
	RESTORE		
	SWAP		
	WINDOW		

1.4.1. Les commandes système

AUTO

Syntaxe : AUTO [<N° de ligne début>,<Incrément>]

Applications : Cette instruction d'aide à la programmation génère automatiquement des numéros de ligne à partir de <N° de ligne début> avec un pas de <Incrément>. Ces deux paramètres peuvent être des nombres entiers ou décimaux.

REMARQUE :

— En cas d'omission des paramètres, AUTO 1ØØ,1Ø s'effectue par défaut.

— Si un numéro de ligne généré est identique à celui d'une ligne de programme déjà en mémoire, un point d'exclamation (!) s'affiche à sa droite, vous indiquant ainsi que l'ancienne ligne sera effacée lors de la validation de la nouvelle.

— L'annulation du mode AUTO s'effectue en tapant RETURN sans entrer de données après le numéro de ligne.

— En donnant à <N° de ligne début> la valeur correspondant à la fin d'un programme déjà en mémoire, on peut compléter celui-ci tout en conservant la numérotation existante.

CLS

Syntaxe : CLS

Application : Efface l'écran, et place le curseur en haut et à gauche.

REMARQUE : CLS efface tout l'écran, quel que soit son contenu, graphique ou texte. Seules PROTECT et WINDOW permettent l'effacement sélectif de données (cf. § 3.1.2 et 3.1.3).

CONT

Syntaxe : CONT

Application : Redémarre un programme interrompu par l'instruction STOP, ou en pressant la touche ESC.

REMARQUE : CONT ne fonctionne que si aucune modification n'a été apportée au programme pendant l'interruption. Voir aussi § 2.6.2.

DEL

Syntaxe : DEL <N° ligne début>[,<N° ligne fin>]

Application : Supprime les lignes comprises entre le numéro de début et le numéro de fin spécifiés.

REMARQUES :

— Si on omet <N° ligne fin>, DEL supprime la ligne correspondant à <N° début>.

— DEL ne s'utilise qu'en mode direct, son emploi dans un programme interrompra l'exécution.

END

Syntaxe : END

Application : Indique la fin d'un programme. Le message "READY !" est affiché à la position du curseur. L'omission de END permet d'éviter l'impression de ce message.

LIST

Syntaxe : LIST <N° de ligne début>[,<N° de ligne fin>]

Application : Liste à l'écran le programme en mémoire, en commençant par <N° ligne début> et s'arrêtant à <N° ligne fin>.

REMARQUES :

— L'omission de <N° ligne fin> permet l'affichage d'une seule ligne de programme.

— Comme pour DEL, l'emploi, de LIST à l'intérieur d'un programme interrompt le déroulement de celui-ci.

— Si les numéros de ligne spécifiés sont plus grands ou plus petits que les numéros existant, le listing se fera à partir des numéros les plus proches en ordre croissant ou décroissant.

NEW

Syntaxe : NEW

Application : Supprime programme et variables de la mémoire du Lynx.

REMARQUE : On peut récupérer un programme accidentellement détruit par NEW en tapant : POKE 26957,192.

RENUM

Syntaxe : RENUM [<N° ligne début>,<Incrément>]

Application : Renumérote toutes les lignes d'un programme. La nouvelle numérotation débute à <N° ligne début> avec un pas de <Incrément>.

REMARQUES :

— En cas d'omission des paramètres, RENUM 1ØØ,1Ø s'effectue par défaut.

— Tous les branchements par GOTO et GOSUB sont également renumérotés.

RUN

Syntaxe : RUN [<N° de ligne>]

Application : Démarre l'exécution du programme en mémoire au numéro de ligne spécifié. Si ce paramètre est omis, le programme débute au plus petit numéro existant.

REMARQUES :

— RUN efface toutes les variables de la mémoire.

— Si <N° de ligne> n'existe pas, une erreur est provoquée.

SPEED

Syntaxe : SPEED N

Application : SPEED règle la vitesse d'exécution d'un programme. N est un entier compris entre Ø et 255. A la mise sous tension, N est initialisé à Ø, ce qui correspond à la rapidité maximale.

REMARQUE : SPEED ne peut s'utiliser qu'en mode programme.

STOP

Syntaxe : STOP

Application : STOP est l'équivalent en mode programme de la touche ESC. Il interrompt l'exécution, et retourne en mode direct en affichant le numéro de la dernière ligne exécutée.

TRACE ON, TRACE OFF

Syntaxe : TRACE ON
TRACE OFF

Applications :TRACE ON liste à l'écran entre <> tous les numéros des lignes exécutées pendant le déroulement d'un programme. Cette instruction fournit une aide précieuse à la mise au point, la détection des erreurs s'effectuant simplement en suivant pas à pas le cheminement du programme. TRACE OFF permet le retour au mode normal.

REMARQUES :

- TRACE peut aussi bien être inséré dans un programme qu'utilisé en mode direct.
- Les numéros de ligne s'affichant à la position courante du curseur qui varie souvent au cours d'un programme, l'écran devient vite illisible en mode TRACE. Pour résoudre ce problème, il suffit de ralentir la vitesse d'affichage avec SPEED, ou d'appuyer de temps à autre sur la touche ESC pour interrompre l'exécution. Le lecteur disposant d'une imprimante pourra également copier le contenu de l'écran pour examen ultérieur. (Voir § 1.4.4).

1.4.2. Le Basic d'usage général

CCHAR

Syntaxe : CCHAR <code 1>*256+<code 2>

Application : Le curseur apparaît à l'écran sous forme d'un pavé clignotant. Ce pavé est constitué de deux caractères affichés alternativement, un espace (code ASCII 32) et un bloc en vidéo inverse (code ASCII 239). On peut modifier ce curseur en changeant le code des caractères affichés par l'instruction CCHAR. <code 1> et <code 2> sont des entiers compris entre 32 et 255.

Exemple :

CCHAR 32*256+42 remplace le bloc en vidéo inverse par un astérisque.

CFR

Syntaxe : CFR N

Application : Modifie la vitesse de clignotement du curseur. N doit être un entier compris entre 1 et 65535.

REMARQUE : Plus la valeur de N est élevée, plus la vitesse de clignotement ralentit. A la mise sous tension N est égal à 5ØØ.

DATA

Syntaxe : DATA <constante>[,<constante>...

Application : DATA permet de stocker une série de constantes numériques ou chaînes, qui pourront être relues et affectées à des variables par READ. <Constante> peut être un nombre, une expression numérique ou une chaîne de caractères. Dans le cas des chaînes, les guillemets doivent être omis.

REMARQUES :

— Une ligne contenant une instruction DATA est ignorée par le Lynx lors de l'exécution du programme, et peut donc être située à n'importe quel point de celui-ci.

— Si plusieurs lignes contiennent des DATA, la lecture avec READ se fera dans l'ordre croissant des numéros de ligne (voir aussi cette instruction).

Exemple :

```
DATA 35,bonjour,1ØØ*PI
```

DIM

Syntaxe 1 : DIM <nom tableau>(<dim.1>[*<dim.2>*<dim.N>])

Applications : Définit les dimensions des tableaux numériques avant leur utilisation selon les règles suivantes :

— Le nombre de dimensions N n'est pas limité.

— <Nom tableau> est un nom de variable numérique standard, il sert ensuite à caractériser le tableau.

— <dim.1>...<dim.N> sont des entiers affectés à chacune des dimensions du tableau pour en définir la taille.

L'emploi des éléments d'un tableau numérique se fait ensuite à l'aide de variables indicées, dont la syntaxe est fonction du nombre de dimensions choisi, et dont les indices sont liés aux valeurs de ces dimensions. Leur

emploi étant assez complexe, voyez aussi les exemples et programmes présentés plus loin pour les méthodes pratiques à mettre en œuvre.

Syntaxe 2 : DIM <Nom de tableau de chaînes>(<longueur>)[(<dim.>)]

Application : Définit la dimension des tableaux de chaînes avant utilisation. (Une seule dimension possible).

— <Nom tableau> est un nom de variable chaîne servant à caractériser le tableau.

— <Longueur> est un entier compris entre Ø et 127, quifixe la longueur maximum, en nombre de caractères, d'une chaîne pouvant être intégrée dans le tableau.

— <dim.> est un entier représentant la taille du tableau, c'est-à-dire le nombre maximum de chaînes qu'il pourra contenir. Si ce paramètre est omis, le tableau sert uniquement à spécifier la longueur d'une chaîne.

Comme pour les tableaux numériques, des variables indicées sont utilisées pour caractériser les éléments d'un tableau de chaînes. Soit par exemple le tableau défini par DIM A$(1Ø) (4), les cinq éléments qui le composent s'appellent A$(Ø), A$(1)...A$(4).

REMARQUES :

— Le nombre d'éléments par dimension n'est limité que par la taille de l'espace mémoire disponible. Sur le Lynx 96 K, DIM A(4ØØØ) est accepté, et laisse 18 K disponibles pour le programme.

— On ne peut pas re-dimensionner un tableau, seuls NEW et RUN le permettent en effaçant toutes les variables de la mémoire.

— Le premier élément d'un tableau a toujours pour indice Ø : DIM A(12) définit donc un tableau à une dimension de 13 éléments, dont le premier est A(Ø) et le dernier A(12).

— La tentative d'utiliser un élément de tableau numérique sans l'avoir préalablement dimensionné ou même sans affecter de valeurs à ses éléments provoque une erreur.

— Un tableau peut porter le même nom qu'une variable simple. L'emploi des minuscules est également autorisé dans le cas des tableaux numériques : A(1), a(1), A, a, A$(12) peuvent coexister en mémoire.

— On peut dimensionner plusieurs tableaux dans une instruction DIM, en séparant leurs noms par des virgules :

DIM A(12),b(11),C$(1Ø) (3Ø),D(5*5*5) est une syntaxe correcte.

— En ce qui concerne les tableaux de chaînes, on peut affecter à un élément du tableau une chaîne de longueur supérieure à celle prévue dans DIM : Les caractères en excès ne sont alors pas pris en compte.

Exemples :

Les tableaux à une dimension :

— DIM A(1Ø) dimensionne un tableau numérique A de 11 cases.

— DIM A$(1Ø) (5) définit un tableau de six chaînes dont la longueur maximum est de 1Ø caractères. Ses éléments seront nommés A$(Ø), A$(1)...A$(5).

— DIM A$(1Ø) crée une chaîne A$, de 1Ø caractères maximum.

Les tableaux numériques multi-dimensionnels

L'emploi de tableaux numériques à plusieurs dimensions est assez complexe sur le Lynx, et de nombreuses méthodes sont envisageables. Un nombre de dimensions supérieur à trois étant rarement rencontré, nous nous limiterons donc à cette valeur dans les exemples proposés.

Tableaux à deux dimensions : DIM A(<dim.1>*<dim.2>)

Un tableau à deux dimensions peut être considéré comme un livre, où <dim.1> représente un numéro de ligne dans une page, et <dim.2> le numéro de la page. Soient L un numéro de ligne et P un numéro de page, on peut adresser chaque élément du tableau avec :

A(L*<dim.2>+P)

L doit être compris entre Ø et <dim.1> non inclus.
P doit être compris entre 1 et <dim.2> inclus.
<dim.1> doit être égal au nombre maximal de lignes désiré + 1.
<dim.2> est égal au nombre de pages désiré.

Cette méthode qui gaspille un peu de mémoire (on n'utilise pas l'élément Ø des tableaux), a cependant le mérite d'être la plus simple à utiliser.

```
100 REM DIM 2
110 CLS
120 T=0,H=5
130 DIM A(5*3)
140  FOR P=1 TO 3
150   FOR L=1 TO 4
160    LET A(L*3+P)=L
170    PRINT@ 3+T,H;"A(";L;",";P;")=";L;
180    LET H=H+10
190   NEXT L
200  LET H=5,T=T+45
210  NEXT P
```

Explications :

12Ø : Initialisation des variables T et H, servant à placer le curseur à l'écran.

13Ø : On dimensionne le tableau A en prévoyant trois pages de quatre lignes.

14Ø,21Ø : Boucle balayant le tableau page par page (ligne 14Ø), puis ligne par ligne (ligne 15Ø), en initialisant chaque ligne avec son numéro (ligne 16Ø). La ligne 17Ø sert à afficher le contenu du tableau (voir également PRINT).

Tableaux à trois dimensions : DIM A(<dim.1>*<dim.2>*<dim.3>)

Si la première dimension représente un numéro de ligne et la deuxième un numéro de page, on peut par analogie considérer la troisième dimension comme un numéro de volume. Soient L un N° de ligne, P un N° de page et V un N° de volume, on peut adresser chaque élément du tableau avec :

A(L*<dim.2>*<dim.3>+P*<dim.3>+V)

L doit être compris entre Ø et <dim.1> exclus.
P doit être compris entre Ø et <dim.2> exclus.
V doit être compris entre 1 et <dim.3> inclus.

Comme précédemment, <dim.1> et <dim.2> correspondent à la taille désirée pour chaque dimension + 1, tandis que <dim.3> est égal à la taille de la dernière dimension.

```
100 REM DIM 3
110 CLS
120 T=0,H=5
130 DIM A(5*5*3)
140  FOR V=1 TO 3
150   FOR P=1 TO 4
160    FOR L=1 TO 4
170    LET A(L*5*3+P*3+V)=L
180    PRINT@ 3+T,H;"A(";L;",";P;",";V;")=";L;
190    LET H=H+10
200    NEXT L
210   NEXT P
220  LET H=5,T=T+45
230  NEXT V
```

Ce programme est identique au précédent, mais comprend trois boucles FOR-NEXT imbriquées au lieu de deux pour balayer la totalité du tableau A( ).

ERROR

Syntaxe: ERROR N

Application: Interrompt le programme et génère une erreur dont le numéro est défini par N. Le message d'erreur correspondant est affiché à l'écran à la position courante du curseur.

REMARQUE: N doit correspondre à l'un des codes d'erreur du Lynx, répertoriés en Appendice.

IF THEN ELSE

Syntaxe: IF <condition 1>[AND / [OR <condition 2>] THEN <instruction>

ELSE <instruction 2>

Applications: La première ligne de programme présentée exécutera l'instruction située après THEN si <condition 1> et/ou <condition 2> sont respectées. Dans le cas contraire, la seconde ligne, facultative, permet d'inclure une autre instruction.

REMARQUE: Le Lynx n'acceptant qu'une seule instruction par ligne, ELSE doit impérativement être placé sur la ligne suivant IF et THEN.

Exemples :

```
- IF A>B AND A<C THEN PRINT "A EST COMPRIS ENTRE B ET C"
```

```
 5 INPUT"DONNEZ UN NOMBRE";A
10 IF A/2=INT(A/2) THEN PRINT "A EST PAIR"
20 ELSE PRINT "A EST IMPAIR"
30 GOTO 5
```

INPUT

Syntaxe : INPUT ["<message>";]<variable>[,<variable>...]

Application : Lorsque une instruction INPUT est rencontrée dans le programme, celui-ci s'arrête momentanément et affiche un point d'interrogation sur l'écran pour indiquer qu'il attend des données. Si un message est inclus dans l'instruction, il est affiché avant le point d'interrogation. <variable> est un nom de variable quelconque, numérique, chaîne ou tableau à laquelle sera affectée la valeur frappée au clavier.

REMARQUE : L'entrée est validée par RETURN.

LET

Syntaxe : [LET]<variable>=<constante>[,<variable>=<constante>...]

Application : Assigne à <variable> la valeur de <constante>. Ces deux paramètres doivent évidemment être du même type, numérique ou chaîne.

REMARQUE : LET peut être omis, et plusieurs affectations de variables effectuées sur une même ligne de programme.

PAUSE

Syntaxe : PAUSE N

Application : Interrompt le programme pour une durée N. Si N=1ØØØØ, l'exécution est suspendue pendant une seconde.

PRINT

Syntaxe : PRINT [@ C,L;][<liste de données>]

Application: Inscrit <liste de données> à l'écran, à la position courante du curseur. Les données à imprimer peuvent être des constantes ou des expressions numériques, des chaînes de caractères ou encore n'importe quel type de variables.

Ces données peuvent être constituées d'un seul ou plusieurs éléments séparés par des ";" ou des ",". Un ";" affiche les éléments les uns à la suite des autres, alors que "," place l'élément qui suit à la position de tabulation suivante (une position toutes les dix colonnes). Si un ";" termine la série de données, le curseur demeure à la fin de celles-ci, alors que dans le cas contraire un passage à la ligne suivante est effectué.

L'option @ permet de placer le curseur, et donc la position d'écriture, à n'importe quel endroit de l'écran. <C> représente le numéro de colonne et <L> le numéro de ligne.

REMARQUES:

— PRINT @ fonctionne en haute-résolution, et autorise donc un placement du curseur hors des positions d'écriture standard. L'écran est divisé en 126 colonnes et 248 lignes pour cette instruction. <C> doit donc être compris entre Ø et 126 ce qui correspond à un pas de deux pixels, et <L> entre Ø et 247, soit un pas de pixel (on appelle pixel le point le plus petit affichable en haute-résolution). Pour respecter les positions d'écriture standard, il faut avoir:

  <C> compris entre 3 et 123 par pas de 3
  <L> compris entre 5 et 245 par pas de 1Ø.

— On peut employer PRINT @ C,L sans ";" pour effacer une ligne de texte à l'écran.

— PRINT peut être remplacé par un point d'interrogation (?) dans un programme.

Exemples:

```
PRINT 4*3,"CHA";"INE"

PRINT@ 3,5;"DEBUT DE L'ECRAN"

PRINT@ 3,235;"DERNIERE LIGNE DE L'ECRAN"
```

```
10 REM  EFFACAGE DE LIGNES PAR PRINT@
20 CLS
25 A=1
30 FOR I=1 TO 800
40 PRINT "A";
50 NEXT I
60 PRINT
70 ?"POUR EFFACER, TAPEZ UNE TOUCHE"
80 X=GETN
90 FOR I=1 TO 20
100 PRINT@ 3,(I*10)-5
110 NEXT I
```

Explications:

3Ø,5Ø : Remplissage de l'écran avec des "A".

8Ø : GETN attend la frappe d'une touche (cf. § 1.5.4).

9Ø,11Ø: Effacement ligne par ligne en positionnant le curseur avec PRINT@.

RANDOM

Syntaxe: RANDOM

Application: Son insertion au début d'un programme permet, lors de l'emploi de nombres aléatoires, d'éviter la répétition d'une même séquence à chaque démarrage par RUN. (Voir aussi RND, § 1.5.1).

READ

Syntaxe: READ <variable>[,<variable>,...]

Application: READ affecte aux variables les constantes contenues dans les listes de DATA. Les variables peuvent être numériques ou chaînes, mais leur type doit correspondre à celui des constantes lues dans les DATA.

REMARQUES:

— Lorsque tous les DATA ont été lus dans un programme, l'emploi de READ provoque un message d'erreur. (Voir RESTORE.)

— Les DATA sont lus dans l'ordre des numéros de ligne, en commençant au début du programme.

REM

Syntaxe: REM <commentaire>

Application: Lorsque l'instruction REM est rencontrée dans un programme, toute information placée à sa droite est considérée comme un commentaire et n'implique aucune exécution.

REMARQUE: REM est utilisé pour clarifier les listings, mais diminue l'espace mémoire disponible.

RESTORE

Syntaxe: RESTORE [<N° de ligne>]

Application: Si <N° de ligne> est omis, RESTORE replace le pointeur des DATA sur la première ligne du programme les contenant. La lecture avec READ se fait ensuite à partir du pointeur. Si une valeur est spécifiée pour <N° de ligne>, le pointeur est placé sur la ligne de DATA ayant ce numéro, ou le suivant immédiatement.

REMARQUE: <N° de ligne> peut être aussi bien un nombre entier que fractionnaire, ou même une expression numérique.

SWAP

Syntaxe: SWAP A,B

Application: Échange le contenu de deux variables. <A> et <B> sont des variables numériques, l'instruction SWAP ne pouvant fonctionner sur les chaînes de caractères.

REMARQUE: SWAP est très utile pour les tris de nombres.

Exemple:

```
100 REM  TRI RIPPLE-SORT D'UNE LISTE DE NOMBRES
110 DIM A(5)
120 DATA 5,12,9,8,4
130 FOR I=0 TO 4
140 READ A(I)
150 NEXT I
160  FOR J=0 TO 4
170   FOR I=J TO 4
180    IF A(J)>A(I) THEN SWAP A(I),A(J)
190   NEXT I
200  PRINT A(J),
210 NEXT J
```

Explications :

110,150: Une liste quelconque de nombre est lue dans un DATA et placée dans le tableau A( ).

160,210: Chaque élément du tableau est successivement comparé à tous les autres, et le plus petit des deux placé en début de liste.

WINDOW

Syntaxe : WINDOW G,D,H,B

Application : L'emploi de WINDOW autorise la réservation d'une zone à l'intérieur de l'écran dont les limites sont définies par <G>, <D>, <H>, <B>, et pour laquelle les instructions d'affichage en mode texte fonctionnene comme s'il s'agissait de la totalité de l'écran. Son contenu n'est pas perturbé par l'exécution de WINDOW.

Les règles d'utilisation de cette instruction sont identiques à celles de PRINT @ : L'écran est divisé en 126 colonnes et 248 lignes.

<G> et <D> correspondent respectivement au bord gauche et au bord droit de la fenêtre, et doivent être compris entre Ø et 126. Les valeurs comprises entre 3 et 123 avec un pas de 3 représentent les positions d'écriture standard.

<H> et <B> correspondent à ses bords supérieur et inférieur et doivent être compris entre Ø et 248. Les valeurs comprises entre 5 et 245 avec un pas de 1Ø accèdent aux 24 lignes d'affichage disponibles.

Bien entendu, <D> et <B> doivent être supérieurs respectivement à <G> et à <H> sinon une erreur est provoquée.

REMARQUES :

— A la mise sous tension, WINDOW est initialisé à WINDOW 3,123,5,245.

— L'instruction a de très nombreuses applications, pour la gestion de l'écran et en mode graphique. (Voir § 3.1.2 et § 4.1.4).

1.4.3. Boucles et branchements

FOR/STEP/NEXT

Syntaxe: FOR N=<valeur initiale> TO <valeur finale>[STEP S]

.

.

<traitement>

.

.

NEXT N

Applications: Ces instructions permettent la création d'une boucle, à l'intérieur de laquelle un traitement donné peut être effectué un certain nombre de fois: Au démarrage, la variable <N> est égale à <valeur initiale>. La section du programme qui suit l'instruction FOR est exécutée jusqu'à rencontre de l'instruction NEXT. La valeur de N est alors incrémentée de <S>, et l'exécution est reprise à la ligne suivant FOR. Lorsque <N> est égal ou supérieur à <valeur finale>, la programme passe à la ligne suivant NEXT. Le nombre de répétitions est donc égal à (1+<valeur finale>–<valeur initiale>)/<S>.

REMARQUES:

— Si l'instruction STEP est omise, <S> est égal à 1 par défaut.

— Après exécution de la boucle, <N> est égal à <valeur finale>+<S>.

— <Valeur initiale> peut être supérieure à <valeur finale>, en spécifiant un pas négatif pour STEP.

— Un FOR doit toujours être suivi de NEXT, sinon une erreur est provoquée.

— On peut imbriquer des boucles entre elles (voir exemple).

— Sur le Lynx, une limitation du Basic interdit la sortie d'une boucle avec un GOTO avant que N ait atteint sa valeur finale. Une tentative de sortie conditionnelle de la boucle (par IF...THEN GOTO) perturbe la pile des GOSUB, ce qui a pour conséquence l'émission d'un message d'erreur au premier RETURN rencontré ! Pour pouvoir inclure des tests à l'intérieur d'une boucle, il est toutefois possible d'utiliser WHILE-WEND ou REPEAT-UNTIL qui n'introduisent aucune perturbation dans les programmes.

Exemple :

```
100 REM  BOUCLE SIMPLE
110 FOR I=1 TO 100
120 PRINT I;
130 NEXT I
```

```
100 REM INCREMENT NEGATIF
110 FOR I=100 TO 1 STEP -1
120 PRINT I;
130 NEXT I
```

```
100 REM  BOUCLES  IMBRIQUEES
110 FOR I=65 TO 90
120 FOR J=1 TO 10
130 PRINT CHR$(I);J
140 NEXT J
150 NEXT I
```

Explication : Les lignes 14Ø et 15Ø ne peuvent être interverties, sous peine de provoquer un message d'erreur.

GOSUB/RETURN

Syntaxe : GOSUB <N° de ligne>

Application : Effectue un branchement vers le sous-programme débutant à <N° de ligne>. Le sous-programme doit se terminer par RETURN, pour permettre un retour au programme principal. L'exécution est alors reprise à la ligne suivant le dernier GOSUB rencontré. Un sous-programme peut être placé n'importe où, mais la recontre d'une instruction RETURN sans que le branchement ait été préalablement effectué par GOSUB provoque une erreur.

REMARQUES :

— Les sous-programmes peuvent en appeler d'autres à leur tour, et leur nombre n'est limité que par la taille de la pile du Lynx (1Ø24 octets).

— <N° de ligne> peut être une expression numérique contenant des variables, pour un branchement calculé.

— L'instruction LABEL examinée plus loin permet de simplifier l'emploi de GOSUB.

— Une sortie intempestive des boucles FOR/NEXT peut perturber le fonctionnement des GOSUB d'un programme (voir FOR/NEXT).

— La multiplication des GOSUB est un moyen simple de structurer les programmes. Elle permet des modifications aisées de l'organisation interne, en changeant des blocs de lignes sans avoir à redéfinir tout le programme, et le réemploi ultérieur de routines fréquemment utilisées sans les réécrire.

Exemples :

```
100 REM  DEMO GOSUB
110 GOSUB 1000
120 PRINT"TERMINE"
130 END
1000 REM SOUS-PROGRAMME BOUCLE
1010 FOR I=1 TO 100
1020 PRINT I;
1030 NEXT I
1040 RETURN
```

Un exemple de programme structuré :

```
100 REM PROGRAMME PRINCIPAL
110 GOSUB LABEL EFFACER
120 GOSUB LABEL REMPLIR
130 GOSUB LABEL COMMENTAIRE
140 PRINT "TERMINE"
150 END
1000 LABEL EFFACER
1010 CLS
1020 RETURN
2000 LABEL REMPLIR
2010 FOR I=1 TO 400
2020 PRINT"A";
2030 NEXT I
2040 RETURN
3000 LABEL COMMENTAIRE
3010 PRINT
3020 PRINT  "TROIS  SOUS  PROGRAMMES SONT UTILISES POUR EFFECTUER
SUCCESSIVEMENT CES OPERATIONS"
3030 RETURN
```

Explication : Le programme principal ne fait qu'appeler successivement tous les sous-programmes, qu'on peut très facilement modifier, remplacer ou même réutiliser sans toucher à la structure générale.

GOTO

Syntaxe : GOTO <N° de ligne>

Application : Effectue un branchement inconditionnel vers la ligne spécifiée. Contrairement à GOSUB, l'emploi de GOTO est en contradiction avec la clarté des listings. Les auteurs conseillent de n'utiliser cette instruction que pour créer des boucles ou lorsque l'emploi d'un sous-programme est impossible.

REMARQUES :

— GOTO est très utile en mode direct pour essayer des parties d'un programme lors de sa mise au point.

— L'instruction LABEL permet de clarifier l'emploi des GOTO.

— Le fait de sortir d'une boucle FOR/NEXT par GOTO perturbe la pile des GOSUB, voir FOR/NEXT.

— Comme pour GOSUB, <N° de ligne> peut être une expression numérique contenant des variables.

Exemples :

```
10 X=GETN
20 IF X=32 THEN GOTO 1000
30 PRINT X
40 GOTO 10
1000 PRINT "TERMINE"
```

Ce petit programme saisit et affiche les codes des touches frappées au clavier, jusqu'à l'entrée d'un espace (voir GETN § 1.5.4).

LABEL

Syntaxe : GOSUB LABEL <nom>
GOTO LABEL <nom>
LABEL <nom>

Application : LABEL permet de remplacer le numéro de ligne dans GOTO et GOSUB par l'étiquette <nom>. Le branchement est ensuite effectué vers la ligne de programme contenant LABEL <nom>. Ce dernier est une chaîne non encadrée de guillemets, et d'une longueur maximum de 8Ø caractères (plus la chaîne est courte, plus la recherche de l'étiquette est rapide).

REMARQUES :

— L'exécution de GOSUB LABEL est plus rapide que le branchement à un numéro de ligne.

— Les espaces sont reconnus par le Basic Lynx comme faisant partie intégrante du nom, attention donc aux erreurs d'entrée !

— Dans le cadre d'une programmation structurée, l'emploi de LABEL rend les sous-programmes indépendants des numéros de lignes, et clarifie les listings.

Exemple :

Reprenons l'exemple de programmation structurée avec GOSUB :

```
100 REM PROGRAMME PRINCIPAL
110 GOSUB 1000
120 GOSUB 2000
130 GOSUB 3000
140 PRINT "TERMINE"
150 END
1000 REM EFFACEMENT ECRAN
1010 CLS
1020 RETURN
2000 REM REMPLISSAGE ECRAN
2010 FOR I=1 TO 400
2020 PRINT"A";
2030 NEXT I
2040 RETURN
3000 REM COMMENTAIRES
3010 PRINT
3020 PRINT  "TROIS  SOUS  PROGRAMMES SONT UTILISES POUR EFFECTUER
SUCCESSIVEMENT CES OPERATIONS"
3030 RETURN
```

PROC/DEFPROC/ENPROC

Syntaxe : PROC <nom> [(<expression 1>,<expression 2>...)]
DEFPROC <nom> [(<variable 1>,<variable 2>...)]
ENDPROC

Application : PROC et DEFPROC sont de puissantes instructions dérivées du langage Pascal, dont la fonction est la création de procédures. Une procédure est très semblable à un sous-programme associé à une étiquette (GOSUB LABEL) : DEFPROC <nom> et ENDPROC marquent respectivement le début de la fin de la procédure, PROC <nom> effectue le branchement à partir du programme principal. <nom> est une chaîne de caractères non entourée de guillemets, et permet de différencier les procédures les unes des autres.

La principale supériorité de DEFPROC sur GOSUB réside dans la possibilité de passer des paramètres au sous-programme, lors de son appel par PROC. Pour ce faire, il suffit d'accoler à DEFPROC une liste de variables séparées par des "," et incluse entre parenthèses. Lors de l'appel de la procédure, ces variables prendront les valeurs spécifiées dans l'instruction PROC effectuant le branchement.

<variables> doivent être des variables numériques.
<expressions> peuvent être des variables, des constantes ou des expressions numériques.

REMARQUES :

— Le nombre d'expressions spécifiées dans PROC doit correspondre au nombre de variables prévues dans DEFPROC.

— <nom> doit toujours être écrit de la même manière pour être reconnu, les espaces étant significatifs.

— Une procédure ne doit jamais être exécutée sans être appelée par PROC, sinon une erreur est provoquée.

— On peut inclure dans la liste d'expressions qui suit PROC une variable, à la condition qu'elle ait été préalablement définie dans le programme : Sa valeur sera transférée à la variable correspondante de DEFPROC. Cette possibilité autorise l'utilisation de variables locales dans un sous-programme, totalement indépendantes de celles du programme principal.

— L'emploi de DEFPROC simplifie l'appel par lui-même d'un sous-programme, les valeurs des variables étant successivement transférées par PROC. Cette possibilité, qu'on nomme récursivité en Pascal, autorise des traitements puissants. Cependant, le Basic Lynx ne possédant pas d'interpréteur récursif gardant en mémoire les variables sur plusieurs niveaux d'appel, l'emploi de procédures récursives reste assez limité.

Exemple :

```
100 REM PROCEDURE
110 CLS
120 INPUT"NOMBRE DE LIGNES A IMPRIMER ";N
130 PRINT
150 PROC BOUCLE (N)
160 PRINT
```

```
170 PRINT"IMPRESSION DES ";N;" LIGNES TERMINEE"
180 PRINT
190 END
200 DEFPROC BOUCLE (A)
210 REPEAT
220 FOR I=1 TO 40
230 PRINT "A";
240 NEXT I
250 LET A=A-1
260 UNTIL A=0
270 ENDPROC
```

Explications :

120 : Entrée par l'utilisateur du nombre de lignes N à imprimer à l'écran.

150 : Appel de la procédure BOUCLE, avec passage de la valeur de N à la variable A de DEFPROC.

200,270 : Procédure affichant A lignes à l'écran.

170 : La valeur de N reste inchangée et peut être réemployée, seule A, variable locale a été modifiée.

```
100 REM PROCEDURE RECURSIVE
110 CLS
120 INPUT"VOTRE NOM ";A$
130 PRINT
150 PROC NOM (LEN(A$))
160 PRINT
170 PRINT"TERMINE"
180 PRINT
190 END
200 DEFPROC NOM (L)
210 PRINT LEFT$(A$,L)
220 L=L-1
230 IF L>0 THEN PROC NOM (L)
240 ENDPROC
```

Explications :

120 : Saisie d'un nom, chargé dans A$.

150 : Appel de la procédure NOM, avec passage de la longueur de A$ à la variable L de DEFPROC.

200,240 : Procédure NOM :

210 : Affichage des L premiers caractères de A$.

22Ø: L est décrémentée de 1.
23Ø: Si L n'est pas égal à Ø, la procédure se rappelle elle-même, avec passage de la nouvelle valeur de L.

REPEAT UNTIL

Syntaxe: REPEAT

.

.

Lignes de programme

.

.

UNTIL <condition 1> [AND / [OR <condition 2>....]

Application: Toutes les lignes de programme situées entre REPEAT et UNTIL sont exécutées en boucle jusqu'à ce que les conditions situées après UNTIL soient vraies (égales à 1).

REMARQUE: Ces commandes sont très proches de WHILE/WEND, avec une différence cependant: Alors que WHILE teste les conditions en début de boucle, UNTIL effectue le test à la fin. Ceci implique que même si la condition est fausse au départ de la boucle, celle-ci sera exécutée au moins une fois.

Exemple:

Programme REPEAT/UNTIL

```
10 I=1
20 REPEAT
30 PRINT "A";
40 I=I+1
50 UNTIL I=800
```

Équivalent FOR/NEXT

```
10 FOR I=1 TO 800
20 PRINT "A";
30 NEXT I
```

REPEAT/UNTIL permet une sortie de boucle assujettie à un ensemble de conditions et à un comptage, tandis que FOR/NEXT n'exécute qu'un comptage. Le programme suivant interrompt la boucle si 2Ø lignes d'écran sont remplies, ou si l'utilisateur appuie sur une touche du clavier (ligne 6Ø).

```
10 I=1
20 REPEAT
30 PRINT "A";
40 I=I+1
50 K=KEYN
60 UNTIL I=800 OR K<>0
70 PRINT
80 PRINT"TERMINE"
```

Si une touche est appuyée immédiatement après RUN, un seul caractère sera imprimé à l'écran.

WHILE/WEND

Syntaxe : WHILE <condition 1> [AND / [OR <condition 2>....]

.
.
Lignes de programme
.
.
WEND

Application : Toutes les lignes de programme situées entre WHILE et WEND sont exécutées en boucle jusqu'à ce que les conditions situées après WHILE soient vraies (=1).

REMARQUE : WHILE teste la condition au début, la boucle n'est donc pas exécutée si les conditions sont fausses avant rencontre de l'instruction.

Exemple :

Programme WHILE/WEND

```
10 I=1
20 REPEAT
30 PRINT "A";
40 I=I+1
50 UNTIL I=800
```

Équivalent FOR/NEXT

```
10 FOR I=1 TO 800
20 PRINT "A";
30 NEXT I
```

Comme pour REPEAT/UNTIL, une série de conditions peuvent être associée au comptage :

```
 50 REM  YOYO
100 CLS
110 C=60,Z=1,K=0
120 WHILE K<>13
130 K=KEYN
140 IF K=32 THEN Z=Z*(-1)
145 PRINT@ C,120;" ";
150 C=C+(Z*3)
160 IF C=123 AND Z=1 THEN C=3
170 IF C=3 AND Z=-1 THEN C=123
180 PRINT@ C,120;"*";
190 WEND
200 PRINT@ 3,200;"TERMINE"
```

Explications :

Ce programme amusant déplace un caractère à l'écran, affiché aux coordonnées C,12Ø. Une pression sur la touche espace change le sens de déplacement en modifiant le signe de l'incrément Z (— 1 ou 1).

Une pression sur la touche RETURN interrompt l'exécution da la boucle.

1.4.4. La gestion des périphériques imprimante et cassette

APPEND

Syntaxe : APPEND "<Nom du programme>"

Application : APPEND permet de charger en mémoire un programme à partir du K7 sans détruire celui qui s'y trouve déjà. Les deux programmes sont automatiquement chaînés l'un à la suite de l'autre.

REMARQUES :

— Il est préférable que les numéros de ligne du programme à charger soient supérieurs à ceux du programme résident. Dans le cas contraire les deux programmes se suivront, mais avec des numéros de ligne identiques et pourront s'exécuter sans provoquer d'erreur ! On peut très bien résoudre ce type de problème en exécutant RENUM une fois l'APPEND terminé. Cette méthode permet le chaînage de tout programme, quelle que soit sa numérotation.

— APPEND est fréquemment employé pour exploiter une bibliothèque de sous-programmes et de procédures, dont la nature les rend de toutes façon indépendants des numéros de ligne.

EXT SPRINT/PPRINT

Syntaxe: EXT SPRINT
EXT PPRINT

Application: Initialise le Basic, pour l'emploi d'une imprimante série ou parallèle. Suivant le type d'interface connecté, SPRINT (Interface série) ou PPRINT (Interface parallèle) est employé. Aucune instruction concernant l'imprimante n'est valide si l'une de ces deux routines n'a été précédemment exécutée.

REMARQUES:

— A la différence des versions 96 K et 128 K, le Lynx 48 K ne possède pas en ROM les instructions nécessaires au pilotage des imprimantes. Le chargement d'un programme sur K7 est indispensable avant toute initialisation. Deux programmes sont prévus, "SPRINT.48K" et "PPRINT.48 K" suivant que l'on utilise un interface série ou parallèle.

— Dans le cas d'un interface série, le Lynx est configuré pour l'imprimante Seikosha GP-25ØX. La connexion d'un autre modèle posant des problèmes de branchement et d'interface, il est plus simple de se connecter sur l'interface parallèle malheureusement fourni en option mais totalement polyvalent.

LINK ON/OFF

Syntaxe: LINK ON
LINK OFF

Application: LINK ON met en parallèle écran et imprimante, toute donnée affichée à l'écran sera immédiatement dupliquée sur l'imprimante ligne par ligne. LINK OFF annule ce mode.

REMARQUES:

— LINK ON ne fonctionne que pour du texte, les graphiques ne sont pas reproduits et les caractères spéciaux sont remplacés par leur code équivalent ASCII.

— Dans la mesure où le Lynx est configuré par la Seikosha GP25Ø, LINK ON ne fonctionne parfaitement qu'avec cette imprimante. Avec une EPSON MX-8Ø par exemple, l'écran est copié en caractères compressés assez peu lisibles.

— Pour éviter la copie du curseur texte sur l'imprimante, il est plus efficace d'employer LINK ON à l'intérieur d'un programme plutôt qu'en mode direct.

LLIST

Syntaxe: LLIST [<N° ligne début>[,<N° ligne fin>]]

Application: Identique à LIST, cette instruction liste sur l'imprimante le programme présent en mémoire entre les deux limites spécifiées.

REMARQUE: L'omission de <N° ligne fin> permet le listage d'une seule ligne.

LOAD

Syntaxe: LOAD "<Nom du programme>"

Application: Charge en mémoire à partir du K7 le programme nommé. Si le programme lu sur le K7 ne correspond pas au nom demandé, le Lynx affiche ce dernier sans effectuer de chargement. Une fois LOAD exécuté, le curseur réapparaît.

REMARQUES:

— LOAD efface de la mémoire programmes et variables.

— La syntaxe LOAD " " permet de visualiser les noms de tous les programmes présents sur une cassette, en faisant défiler celle-ci depuis le début.

— Si la sauvegarde du programme a été effectuée en incluant un numéro de ligne (voir SAVE), il s'exécutera à partir de ce numéro une fois le chargement terminé.

— Dans le cas où la fiche "Remote" du cordon de liaison K7 est branchée, LOAD démarre et arrête automatiquement le défilement. Toutefois, cette fiche ne fonctionne pas avec tous les magnétophones du commerce.

LPRINT

Syntaxe: LPRINT [<liste de données>]

Application: Cette instruction est totalement équivalente à PRINT pour

l'imprimante. Les séparateurs "," ou ";" fonctionnent de manière identique (voir § 1.4.2).

REMARQUES :

— LPRINT sans données provoque un passage à la ligne.

— TAB et @ ne sont pas valides avec LPRINT.

MLOAD

Syntaxe : MLOAD "<Nom du programme>"

Application : D'emploi similaire à LOAD, MLOAD charge une zone mémoire ou un programme en langage-machine à partir du K7. Programme et variables ne sont pas affectés par MLOAD.

REMARQUE : Le chargement s'effectue dans la même zone-mémoire que celle spécifiée lors de la sauvegarde. Celle-ci s'effectue à l'aide de la commande D du moniteur, et ne peut donc se faire à partir du Basic. (Voir § 2.5.)

SAVE

Syntaxe : SAVE "<Nom du programme>"[,<N° de ligne>]

Application : Sauvegarde le programme nommé sur K7. Si la fiche "Remote" est connectée, SAVE démarre et arrête le magnétophone.

REMARQUES :

— <Nom du programme> peut avoir une longueur maximum de 8Ø caractères.

— Si l'option <N° de ligne> est utilisée, le programme s'exécutera automatiquement à partir de la ligne spécifiée après son chargement par LOAD.

TAPE

Syntaxe : TAPE N

Application : Règle la vitesse de sauvegarde et de chargement des programmes par SAVE et LOAD. N est un entier compris entre Ø et 5, et correspond aux six vitesses disponibles (un baud=un bit/seconde) :

Valeur de N	*Vitesse en Baud*
Ø	6ØØ
1	9ØØ
2	12ØØ
3	15ØØ
4	18ØØ
5	21ØØ

REMARQUES :

— TAPE Ø est la valeur par défaut à la mise sous tension ; cette vitesse de transfert assez faible assure une grande fiabilité. TAPE 2 se révèle être un bon compromis pour les applications courantes.

— Un programme doit être impérativement rechargé à la même vitesse que lors de sa sauvegarde.

VERIFY

Syntaxe : VERIFY "<Nom du programme>"

Application : Cette commande très utile autorise la vérification du bon enregistrement d'un programme sur K7 après exécution de SAVE. Si une erreur est décelée, le Lynx affiche "BAD TAPE" à l'écran au lieu de l'habituel curseur.

1.5. Les fonctions du Basic

A la différence d'une instruction, une fonction s'utilise toujours en conjonction avec une autre commande du Basic, en général PRINT ou LET. Par exemple : LET A=VAL (A$).

Les syntaxes données pour les fonctions suivantes doivent en conséquence être considérées comme incomplètes par le lecteur.

Comme pour les Instructions, les Fonctions sont regroupées selon quatre catégories :

1.5.1 *Usage général*	1.5.2 *Traitement nombres*	1.5.3 *Traitement chaînes*	1.5.4 *Saisies*
FALSE POS RAND RND TAB TRUE VPOS	ABS ANTILOG ARCCOS ARCSIN ARCTAN Conversions COS DEG DIV EXP FACT FRAC INF INT LOG LN MOD PI RAD ROUND SIN SGN SQR TAN TRAIL	ASC CHR$ LEFT$ LEN MID$ RIGHT$ STR$ UPC$ VAL	GETN GET$ KEYN KEY$

1.5.1. Les fonctions d'usage général

FALSE/TRUE

Syntaxe: FALSE
TRUE

Application: Ce sont deux variables réservées du Basic, au même titre que PI. FALSE (Faux) vaut Ø et TRUE (Vrai) vaut 1. On les utilise dans les comparaisons avec IF, pour améliorer la lisibilité des programmes entre autres.

Exemple :

IF (A>2 AND B>1Ø)=TRUE THEN....<Instruction>

Si les conditions placées entre parenthèses sont vraies, l'expression complète prend la valeur 1, et est donc égale à TRUE.

POS

Syntaxe : POS

Application : Retourne la position horizontale du curseur, donc une valeur allant de Ø à 126. Les positions d'écriture standard vont de 3 à 123 par pas de 3 (voir PRINT @ , § 1.4.2.).

REMARQUE : Cette fonction est inutilisable en mode direct, car après un RETURN, le curseur retourne à la première colonne de la ligne suivante, et POS renvoie toujours 3.

RAND

Syntaxe : RAND (N)

Application : Renvoie un nombre entier pseudo-aléatoire, compris entre Ø et N−1. N doit être un entier compris entre Ø et 65535.

REMARQUES :

— RAND (N)+1 permet l'obtention d'un entier compris entre 1 et N.

— Voir également l'instruction RANDOM, § 1.4.2.

Exemple :

```
100 REM  JEU DE DES
110 CLS
120 PRINT@ 3,120;"POUR JOUER TAPEZ UNE TOUCHE";
130 X=GETN
140 PRINT@ 87,120;RAND(6)+1
150 GOTO 120
```

RND

Syntaxe : RND

Application : Renvoie un nombre réel pseudo-aléatoire compris entre Ø et 1, 1 étant exclus.

TAB

Syntaxe : PRINT TAB N;[<liste de données>]

Application : TAB autorise le positionnement horizontal du curseur sur n'importe laquelle des 4Ø positions standard d'écriture, sans passer par PRINT@. N est un entier positif représentant le numéro de colonne. Si N>39, le curseur saute N DIV 4Ø lignes et se place à la colonne calculée par N MOD 4Ø.

REMARQUES :

— Le ";" est obligatoire après TAB.

— TAB peut être inclus à n'importe quel endroit d'une instruction PRINT.

Exemple :

```
PRINT "CHAINE";TAB 15;"DE CARACTERES"
```

VPOS

Syntaxe : VPOS

Application : Retourne la position verticale du curseur, donc une valeur allant de Ø à 248. Les positions d'écriture standard vont de 5 à 245 par pas de 1Ø (voir PRINT@, § 1.4.2).

REMARQUE: VPOS associé à POS autorise la lecture de la position du curseur à l'écran dans ses deux axes en mode texte.

1.5.2. Le traitement numérique

ABS

Syntaxe : ABS(A)

Application : Renvoie la valeur absolue de A.

Exemple :

ABS(1Ø) renvoie 1Ø, ABS(−1Ø) renvoie 1Ø également.

ANTILOG

Syntaxe : ANTILOG(A)

Application : Calcule l'inverse du logarithme décimal de A, c'est-à-dire l'exponentielle en base 1Ø de A (1Ø**A).

ARCOS

Syntaxe : ARCCOS(A)

Application : ARCOS calcule l'angle en radians, dont A est le Cosinus.

ARCSIN

Syntaxe : ARCSIN(A)

Application : ARCSIN calcule l'angle en radians, dont A est le Sinus.

ARCTAN

Syntaxe : ARCTAN(A)

Application : ARCTAN calcule l'angle en radians, dont A est la tangente.

REMARQUE : Les trois fonctions précédentes sont les fonctions inverses de COS, SIN et TAN respectivement.

Conversions numériques

Syntaxes : **#**D
&H
BIN(B)

Applications :

— **#** convertit le nombre décimal D en son équivalent hexadécimal. D doit être un entier positif inférieur à 65536.

— **&** convertit le nombre hexadécimal H en son équivalent décimal.

— **BIN** convertit le nombre binaire B en son équivalent décimal. Un nombre binaire comprend huit chiffres au maximum et est de la forme : 1Ø1Ø1ØØ1.

REMARQUE: On peut utiliser directement des nombres hexadécimaux ou binaires dans un programme ou une expression numérique en les faisant précéder du préfixe de conversion: Le Basic les considère dans ce cas comme leurs équivalents décimaux.

Exemple:

```
1Ø D=&FF,B=BIN(11111111)
2Ø PRINT D,B,#D
3Ø END
```

COS

Syntaxe: COS(A)

Application: Renvoie le cosinus de l'angle A, qui doit être exprimé en Radians.

DEG

Syntaxe: DEG(<angle>)

Application: Convertit en degrés la valeur d'un angle exprimé en Radians.

Exemple:

```
PRINT DEG(ARCSIN(1))
```

DIV

Syntaxe: A DIV B

Application: DIV est en fait un opérateur arithmétique qui renvoie le résultat de la division entière de A par B, c'est-à-dire INT(A/B).

REMARQUE: On utilise beaucoup DIV dans les calculs d'adresses sur deux octets: Cette fonction permet de connaître directement l'octet de poids fort correspondant à l'adresse N dans N DIV 256 (voir également MOD et § 2.2).

EXP

Syntaxe: EXP(A)

Application : Calcule l'exponentielle de A en base e, c'est-à-dire e à la puissance A (e**A).

REMARQUE: EXP est la fonction inverse de LN.

FACT

Syntaxe : FACT(A)

Application : Retourne la factorielle de A, soit en notation standard A !

REMARQUE: Si A n'est pas un entier, FACT travaille sur la partie entière uniquement.

FRAC

Syntaxe : FRAC(A)

Application : Cette fonction originale extrait la partie fractionnelle d'un nombre. Si A est un entier, FRAC(A) vaut Ø.

INF

Syntaxe : INF

Application : INF est une constante, égale à 9.99 E+63. C'est le nombre le plus grand que peut traiter le Lynx.

REMARQUE: INF ne sert pratiquement à rien !

INT

Syntaxe : INT(A)

Application : INT retourne la partie entière de A. L'arrondi n'est pas exact, dans la mesure où l'entier inférieur est toujours renvoyé (voir § 1.7.2). Si A est une fraction inférieure à 1, INT(A) vaut Ø.

LOG

Syntaxe : LOG(A)

Application : Calcule le logarithme en base 1Ø du nombre A.

LN

Syntaxe: LN(A)

Application: Calcule le logarithme Népérien du nombre A.

MOD

Syntaxe: A MOD B

Application: MOD est comme DIV un opérateur arithmétique, il renvoie le reste de la division entière de A par B, c'est-à-dire: A—((A DIV B)*B).

REMARQUE: MOD sert également dans les calculs d'adresses sur deux octets, le résultat de N MOD 256 correspond à l'octet de poids faible de l'adresse N (voir DIV et § 2.2).

PI

Syntaxe: PI

Application: La constante PI est stockée en ROM sur sept décimales.

RAD

Syntaxe: RAD(<angle>)

Application: Convertit en Radians la valeur d'un angle exprimé en degrés.

Exemple:

```
PRINT SIN(RAD(9Ø))
```

ROUND ON/OFF

Syntaxe: ROUND ON
ROUND OFF

Application: Le Lynx effectue les calculs sur sept décimales, et affiche les résultats en les arrondissant sur cinq décimales. C'est le mode ROUND ON. En spécifiant ROUND OFF, on obtient l'affichage des résultats en 7 décimales.

SIN

Syntaxe : SIN(A)

Application : Calcule le sinus de l'angle A, celui-ci étant exprimé en Radians.

SGN

Syntaxe : SGN(A)

Application : Retourne un nombre en fonction du signe de A :

— Si A<Ø, SGN(A) est égal à −1.
— Si A>Ø, SGN(A) vaut 1.
— Si A=Ø, SGN(A)=Ø.

SQR

Syntaxe : SQR(N)

Application : SQR calcule la racine carrée de N.

REMARQUE : Cette fonction est l'inverse de N**2.

TAN

Syntaxe : TAN(A)

Application : Calcule la tangente de l'angle A, celui-ci étant exprimé en Radians.

TRAIL ON/OFF

Syntaxe : TRAIL ON
TRAIL OFF

Application : TRAIL ON ajoute des "Ø" à la droite d'un nombre affiché, dans le format imposé par ROUND (5 ou 7 décimales). TRAIL OFF annue ce mode.

1.5.3. Le traitement des chaînes de caractères

ASC

Syntaxe: ASC(A$)

Application: ASC renvoie le code ASCII du premier caractère de la chaîne A$. Si A$ est une chaîne vide, ASC(A$) est égal à 13 ce qui correspond au code "Retour chariot" (touche RETURN). Si A$ n'a pas été préalablement définie, une erreur est provoquée.

REMARQUE: ASC est la fonction inverse de CHR$.

CHR$

Syntaxe: CHR$(A)

Application: CHR$ renvoie le caractère de code ASCII A. L'argument peut être un entier ou une expression numérique contenant des variables préalablement définies.

REMARQUES:

— S'emploie aussi bien pour afficher des caractères que pour envoyer des codes spéciaux à l'écran ou à l'imprimante (voir § 3.1.3).

— Les caractères du jeu ASCII standard ont des codes compris entre 32 et 127. Les mosaïques graphiques ont des codes compris entre 224 et 249 (voir § 3.2).

LEFT$

Syntaxe: LEFT$(A$, N)

Application: LEFT$ extrait une sous-chaîne de A$ constituée des N premiers caractères à gauche de A$. N doit être un entier positif ou une expression numérique équivalente.

REMARQUES:

— Si N=Ø ou si A$=" " ou si N est égal à la longueur totale de A$ plus 1, LEFT$ renvoie une chaîne vide.

— Si N est supérieur à la longueur de A$ plus 1, une erreur est provoquée.

Exemple :

```
10 INPUT "DONNEZ VOTRE NOM ";A$
20 FOR I=1 TO LEN(A$)
30 PRINT LEFT$(A$,I)
40 NEXT I
```

LEN

Syntaxe : LEN(A$)

Application : Calcule la longueur de la chaîne A$, en nombre de caractères. Si A$=" ", alors LEN(A$)=Ø.

REMARQUE : A$ doit être une chaîne préalablement définie.

Exemple :

```
10 INPUT "DONNEZ UN MOT DE CINQ LETTRES ";A$
20 IF LEN(A$)=5 THEN GOTO 50
30 PRINT "ERREUR, CINQ LETTRES SVP"
40 GOTO 10
50 PRINT A$
60 GOTO 10
```

MID$

Syntaxe : MID$(A$, N, M)

Application : MID$ est la plus puissante des fonctions de traitement des chaînes du Lynx. Elle autorise l'extraction de n'importe quelle partie d'une chaîne de caractères : La chaîne retournée contiendra M caractères de A$, en commençant par le N^{e}.

REMARQUES :

— Si N est supérieur à la longueur de A$ ou inférieur à 1, MID$ renvoie une chaîne vide.

— Si M est supérieur au nombre de caractères compris entre le N^{e} et la fin de A$, MID$ renvoie la totalité de ceux-ci.

Exemples :

```
10 A$="123456789"
20 PRINT MID$(A$,2,5)
```

Nous imprimera : 23456

Voici, pour démontrer les possibilités de MID$, un petit programme qui utilise cette fonction de manière amusante.

Exemples :

```
100 REM  DEFILEMENT TEXTE
110 CLS
120 DIM A$(120)
130 A$="ECRAN UTILISANT LES POSSIBILITES DE MID$ CECI EST UN
DEFILEMENT DE TEXTE A L'ECRAN UTILISANT LES POSSIBILITES DE
MID$  "
140 P=41
150 PRINT@ 3,120;
160 PRINT MID$(A$,P,40);
170 P=P+1
180 IF P=LEN(A$)-41 THEN LET P=1
190 GOTO 150
```

Explications :

Le principe du programme est assez simple : On crée une longue chaîne, puis on en extrait une sous-chaîne qui est affichée toujours au même endroit de l'écran. En décalant la sous-chaîne d'un cran vers la droite à chaque impression, on obtient un effet de défilement.

12Ø : Dimensionne la chaîne A$.

13Ø : Création d'une chaîne A$ contenant le texte à imprimer.

15Ø : Positionnement du curseur au centre de l'écran.

16Ø : Impression de 4Ø caractères de A$, à partir du P^e.

17Ø : Incrément de P.

18Ø : Lorsque toute la chaîne a été lue, on réinitialise P à 1.

En modifiant A$ et la ligne 15Ø, ce programme pourra être employé par le lecteur dans d'autres applications.

RIGHT$

Syntaxe : RIGHT$(A$, N)

Application : RIGHT$ extrait une sous-chaîne de A$ constituée des N derniers caractères à droite de A$. N doit être un entier positif ou une expression numérique équivalente.

REMARQUES :

— Si N=Ø ou si A$=" " ou si N est égal à la longueur totale de A$ plus 1, RIGHT$ renvoie une chaîne vide.

— Si N est supérieur à la longueur de A$ plus 1, une erreur est provoquée.

STR$

Syntaxe : STR$(N)

Application : Convertit une constante ou expression numérique en chaîne de caractères.

REMARQUES :

— STR$ est la fonction inverse de VAL.

— Cette fonction s'utilise pour effectuer sur des nombres des traitements réservés aux chaînes. VAL permet la conversion inverse.

Exemples :

```
10 A=8.584,B=3.999
20 A$=STR$(A),B$=STR$(B)
30 A$=LEFT$(A$,3),B$=LEFT$(B$,3)
40 A=VAL(A$),B=VAL(B$)
50 PRINT"A ET B SUR UNE SEULE DECIMALE ";A,B
```

Explication :

2Ø : A et B sont convertis en A$ et B$.

3Ø : Prise en compte des trois premiers caractères de A$ et B$.

4Ø : On reconvertit A$ et B$ en variables numériques.

UPC$

Syntaxe : UPC$(A$)

Application : Cette fonction, spécifique au Basic Lynx, convertit tous les caractères minuscules de A$ en majuscules. Les nombres et symboles restent inchangés.

REMARQUE : UPC$ est très pratique pour la saisie de données dans les fichiers de noms et d'adresses, la conversion en majuscules permettant la standardisation des entrées.

Exemples :

```
100 REM CONVERSION EN MAJUSCULES
110 CLS
120 PRINT"ENTREZ DU TEXTE, MAJ. OU MIN."
130 PRINT
140 PRINT"POUR ARRETER, TAPER <DELETE>"
150 PRINT
160 WHILE ASC(G$)<>8
170 G$=GET$
180 PRINT UPC$(G$);
190 WEND
200 PRINT
210 PRINT"TERMINE"
```

VAL

Syntaxe : VAL(A$)

Application : Retourne l'équivalent numérique d'une chaîne de caractères :

— Si A$ contient un nombre, VAL renvoie ce nombre.

— Si A$ contient une expression numérique ou une chaîne, VAL ne peut effectuer le calcul et renvoie Ø.

— Si A$ est composé de nombres et de caractères, VAL ne renverra le nombre que s'il est placé au début de A$.

REMARQUE : VAL peut fonctionner pour des nombres hexadécimaux précédés de "&".

Exemples :

```
VAL("5 Francs")=5
VAL("X=3*4")=0
VAL("BONJOUR")=0
VAL("6/12")=6
```

1.5.4. La saisie clavier

Les Fonctions de saisie clavier sont les seules commandes du Basic autorisant une intéraction directe entre l'utilisateur et l'ordinateur par l'intermédiaire du clavier. Ce chapitre n'aborde que la saisie en Basic, une méthode utilisant le langage-machine est proposée au paragraphe 4.1.5.

GETN/GET$

Syntaxe 1 : X=GETN

Application : Interrompt le déroulement du programme et attend l'appui d'une touche au clavier. Dès que cela est fait, le code de la touche pressée est stocké dans la variable X.

Syntaxe 2 : X$=GET$

Application : Identique à GETN, mais stocke dans X$ le caractère de la touche pressée au lieu de son code.

KEYN/KEY$

Syntaxe 1 : K=KEYN

Application : KEYN lit le clavier sans interrompre l'exécution du programme. Si une touche est pressée lorsque l'instruction est rencontrée, son code est stocké dans la variable K. Si aucune touche n'est actionnée, K vaut Ø.

Syntaxe 2 : K$=KEY$

Application : KEY$ est identique à KEYN, mais stocke le caractère de la touche pressée dans X$, qui sera une chaîne vide si aucune touche n'est utilisée.

REMARQUES :

— KEYN et KEY$ sont les fonctions de saisie privilégiées pour les jeux en temps réel : Par exemple, la détection d'une action sur les flèches du clavier pour déplacer un mobile à l'écran.

— Ces fonctions ne prennent pas en compte l'autorépétition des touches du clavier. En conséquence, le fait de laisser le doigt appuyé sur une touche lors d'une saisie par KEYN ou KEY$ n'est pas interprété correctement dans un programme : Il faut relâcher le doigt et réappuyer ! !

Le programme suivant ne peut donc fonctionner dans un jeu en temps réel :

```
10 K=KEYN
20 IF K=0 THEN GOTO 10
30 REM INSERER PROGRAMME
.
.
200 GOTO 10
```

On dispose de deux moyens pour tourner cette difficulté :

— Le premier consiste à démarrer un faux "repeat" par programme lorsque la touche est appuyée. Une deuxième pression sur celle-ci annule ce mode. C'est cette méthode qui est employée dans le programme "TENNIS", cité en exemple plus loin.

```
100 REM FAUX REPEAT
105 K=0,M=0
110 K=KEYN
120 IF K=M THEN LET M=0,K=0
130 IF NOT K=0 THEN LET M=K
140 IF K=0 THEN LET K=M
150 REM INSERER PROGRAMME
.
.
.
1000 GOTO 110
```

Explication :

1Ø5 : Initialisation de K et M. K correspond au code de la touche pressée, et M mémorise le code précédent.

11Ø : Lecture du clavier.

12Ø : Si le code de la touche pressée correspond au code mémorisé dans M, il s'agit d'une deuxième pression sur la même touche et le "repeat" est annulé en initialisant K et M à Ø.

13Ø : Si une touche est pressée, on mémorise son code en M.

14Ø : Si aucune touche n'est pressée, on simule une pression sur la touche dont le code est mémorisé dans M, en chargeant M dans K.

15Ø : Ici, il faut insérer le programme proprement dit. La valeur de K correspond au code de la touche actionnée, ou à Ø.

1ØØØ: Retour à la saisie.

— Un deuxième moyen est possible, par l'intermédiaire du langage-machine. La lecture de certains ports du Z8Ø autorise une saisie clavier fugitive, avec prise en compte de l'autorépétition des touches. La procédure est détaillée au paragraphe 4.1.5.

Exemple : Le programme TENNIS :

```
100 REM TENNIS
110 REM
120 REM TERRAIN DE JEU
150 TEXT
160 WINDOW 3,123,205,245
180 FOR I=3 TO 117 STEP 3
190 PRINT@ I,5;CHR$(236);
200 PRINT@ I,195;CHR$(227);
210 NEXT I
220 FOR I=15 TO 185 STEP 10
230 PRINT@ 120,I;CHR$(229);
240 NEXT I
250 PRINT@ 120,5;CHR$(228);
260 PRINT@ 120,195;CHR$(225);
270 X=3,Y=105
280 PRINT@ X,Y;CHR$(239);
290 GOSUB 570
300 PRINT@ x,y;CHR$(240);
310 REM BOUCLE DE JEU
320 M=0,K=0
330 K=KEYN
340 IF K=M THEN LET M=0,K=0
350 IF NOT K=0 THEN LET M=K
360 ELSE LET K=M
370 IF K=10 OR K=11 THEN PRINT@ X,Y;CHR$(224);
380 IF K=11 AND NOT Y=15 THEN LET Y=Y-10
390 IF K=10 AND NOT Y=185 THEN LET Y=Y+10
400 PRINT@ X,Y;CHR$(239);
410 IF y=15 THEN LET G=1
420 IF y=185 THEN LET G=-1
430 IF x=117 AND F=1 THEN LET F=-1
460 IF x=6 AND y=Y THEN LET F=1,G=0
470 IF x=6 AND y=Y+10 THEN LET F=1,G=1
480 IF x=6 AND y=Y-10 THEN LET F=1,G=-1
490 PRINT@ x,y;CHR$(224);
500 x=x+(F*3)
510 y=y+(G*10)
520 IF x=117 OR y=15 OR y=185 THEN GOSUB 620
530 PRINT@ x,y;CHR$(240);
540 IF x=3 THEN GOTO 640
550 IF x=6 AND y=Y OR x=6 AND y=Y+10 OR x=6 AND y=Y-10 THEN GOSUB 620
560 GOTO 330
570 REM  SERVICE
580 IF RND>=0.5 THEN LET G=1
590 ELSE LET G=-1
600 LET x=3,y=(10*RAND(18))+5,F=1
610 RETURN
620 BEEP 3000,1,63
630 RETURN
640 REM FIN DE JEU
```

```
650 FOR I=500 TO 50 STEP -10
660 BEEP I,2,63
670 NEXT I
680 PRINT@ 3,205;"VOUS AVEZ PERDU, POUR REJOUER TAPEZ <R>"
690 R$=GET$
700 IF R$="R" THEN GOTO 100
710 GOTO 690
```

Explication :

12Ø,28Ø : Dessin de l'aire de jeu à l'aide de caractères semi-graphiques ; affichage de la raquette aux coordonnées X et Y.

29Ø,3ØØ : Appel du sous-programme "SERVICE", et affichage de la balle (coordonnées x,y).

31Ø : Boucle principale incluant :

32Ø,36Ø : Le programme de saisie clavier présenté avec KEYN.

37Ø,4ØØ : La valeur de K (code de la touche pressée) est utilisée pour déplacer la raquette si ↑ ou ↓ est actionnée.

41Ø,48Ø : Si la balle atteint l'un des bords du terrain de jeux, sa direction est modifiée et mémorisée dans F et G.

49Ø : Effacement de la balle.

5ØØ,51Ø : Calculs des nouvelles coordonnées de la balle en fonction de la direction définie par F et G.

52Ø : Appel du sous-programme "SON" si le bord de l'écran est atteint par la balle.

53Ø : Réaffichage de la balle.

54Ø : Fin du jeu si la balle est perdue (lorsqu'elle dépasse la raquette).

55Ø : Génération d'un son lorsque la balle touche la raquette.

56Ø : Retour à la saisie clavier.

57Ø,61Ø : Sous-programme service, envoie la balle à partir d'une position aléatoire et lui donne une direction, définie par F et G, également aléatoire.

64Ø,72Ø : Sortie du jeu.

1.6. Instructions et fonctions non résidentes

Malgré la richesse de son "vocabulaire", le Lynx n'est cependant pas parfait et certaines commandes disponibles sur les Basic les plus évolués lui font défaut. Nous avons essayé de sélectionner celles d'entre elles qui nous paraissaient les plus importantes, et dont la simulation par un petit programme reste dans les limites de complexité raisonnables.

D'autres possibilités, utilisant le Moniteur et le langage-machine, seront vues dans les sections suivantes.

1.6.1. Instructions absentes du Basic Lynx

CLEAR

Application: Efface toutes les variables et tableaux de la mémoire sans effacer le programme.

Equivalent Lynx: Seul RUN permet l'effacement des variables. Le plus simple est d'inclure dans un programme, à l'endroit où l'on a besoin d'un CLEAR, l'instruction GOTO LABEL CLEAR puis:

```
65500 LABEL CLEAR
65501 RUN N
```

Avec N=Numéro de la ligne suivant l'instruction GOTO LABEL.

MID$(A$, N [,M])=B$

Application: Cette instruction, inverse de la fonction du même nom, permet de remplacer M caractères dans A$, à partir du N^{e}, par les M premiers caractères de B$. Si M est omis, B$ est inséré en entier. Si B$ contient plus de caractères que A$ ne peut en recevoir, les caractères excédentaires sont ignorés.

Equivalent Lynx: Le Lynx possède bien la fonction MID$, permettant de lire une fraction d'une chaîne, mais pas l'instruction effectuant le remplacement d'une partie d'un chaîne par une autre. L'expression suivante la remplace:

```
A$=LEFT$(A$,N-1)+LEFT$(B$,M)+RIGHT$(A$,LEN(A$)-(N+M-1))
```

PRINT USING "<chaîne>";<liste de données>

Application: Affiche des nombres selon un format défini par <chaîne>. Dans le cas de tableaux numériques, et surtout lorsque ceux-ci comportent des décimales, un affichage selon un format identique (alignement sur le point décimal, cadrage à droite) améliore grandement la lisibilité et la présentation.

Equivalent Lynx: Les vrais PRINT USING autorisent une grande liberté dans le choix des formats d'impression. Néanmoins, le programme qui suit effectue toutes les opérations d'arrondi exact et de formattage d'un nombre: Celui-ci sera justifié à droite, avec un espace accolé à la dernière décimale s'il est positif, et le signe "−" s'il est négatif, selon la norme de cette instruction. Si le nombre est trop grand pour le format il sera remplacé par des "**".

REMARQUES:

— On appelle le programme par GOSUB LABEL PRINTUS.

— Les variables que vous devez passer au sous-programme dont N, le nombre à traiter et P$ qui détermine le format. P$ est une chaîne de caractères de la forme: P$="***.***". Le nombre d'astérisques employé avant et après le point décimal détermine le nombre de chiffres avant et après la virgule.

— Une fois le programme exécuté, N$ contient le nombre formatté à imprimer sous la forme d'une chaîne de caractères.

```
60000 LABEL PRINTUS
60010 D=0,Z=0,Z$="0000000000",C$="                    ",S$="********  *
60020 FOR P=1 TO LEN(P$)
60030 IF MID$(P$,P,1)="." THEN LET Z=P
60040 NEXT P
60050 IF Z<>0 THEN LET P=Z,D=LEN(P$)-P
60060 E=P-1
60070 N=INT((N*10**D)+0.5)/10**D,N$=STR$(N)
60080 IF SGN(N)=1 THEN LET N$=" "+N$
60090 M$=LEFT$(N$,1),N$=RIGHT$(N$,LEN(N$)-1)
60100 Z=0
60110 FOR P=1 TO LEN(N$)
60120 IF MID$(N$,P,1)="." THEN LET Z=P
60130 NEXT P
60140 IF Z<>0 THEN LET P=Z
60150 E$=LEFT$(N$,P-1)
60160 IF P>LEN(N$) THEN LET D$=""
60170 ELSE LET D$=RIGHT$(N$,LEN(N$)-P)
60180 L=LEN(D$)
```

```
60190 IF L<D THEN LET D$=D$+LEFT$(Z$,D-L)
60200 L=LEN(E$)
60210 IF L<E THEN LET E$=LEFT$(C$,E-L)+E$
60220 IF (D$="")=0 THEN GOTO 60250
60230 N$=E$+M$
60240 GOTO 60260
60250 N$=E$+"."+D$+M$
60260 IF LEN(N$)>E+D+2 THEN LET N$=LEFT$(S$,E+D+1)
60270 RETURN
```

Explicatons:

60010	:	Initialisation des variables.
60020,60040	:	Lecture du format P$, et mémorisation de la position du point décimal dans Z.
60050,60060	:	Calcul de la longueur de la partie entière (E) et de la partie décimale (D).
60070	:	N est arrondi au format désiré et stocké dans N$.
60080,60090	:	Mémorisation du signe de N dans M$.
60110,60150	:	Stockage dans E$ de la partie entière de N$.
60160,60170	:	Stockage dans D$ de la partie décimale de N$.
60180,60190	:	Ajoute des "0" à droite de la partie décimale, pour la faire correspondre au format.
60200,60210	:	De même, on ajoute des espaces à gauche de la partie entière.
60220,60250	:	Reconstitution de N$, avec ajout du signe "–" le cas échéant.
60260	:	En cas de dépassement de format, on remplace le contenu de N$ par des astérisques.

REMARQUE: Le programme est rallongé par l'impossibilité de grouper plusieurs instructions sur une même ligne, ainsi que par celle de sortir des boucles FOR/NEXT à l'aide de GOTO.

1.6.2. Fonctions non résidentes

Arrondi exact

Nous savons que l'arrondi obtenu par la fonction INT n'est pas exact, puisqu'elle retourne toujours l'entier inférieur. Voici la formule permettant d'obtenir l'arrondi pour X, D étant le nombre de décimales souhaité :

INT((X*1Ø**D)+Ø.5)/1Ø**D

Si on veut convertir un nombre à l'entier le plus proche, l'expression se réduit à :

INT(X+Ø.5)

INPUT$(N)

Application : Équivalent de l'instruction GET$, avec la différence qu'au lieu d'un seul, N caractères sont saisis au clavier.

Equivalent Lynx :

```
10 FOR I=1 TO N
20 X$=GET$
30 NEXT I
```

INSTR(I, A$, B$)

Application : Recherche la chaîne B$ à l'intérieur de A$ à partir de la position I. La valeur retournée est Ø si B$ n'est pas trouvé, ou la position du premier caractère de B$ dans A$.

Equivalent Lynx : Un très court programme remplacera cette fonction indispensable. Il pourra être incorporé très aisément dans un ensemble plus important.

REMARQUES :

— A l'appel du sous-programme par GOSUB LABEL INSTR, A$, B$ et I doivent être définis.

— Après exécution, P contiendra la position de B$ dans A$, ou Ø si la recherche est négative.

```
60000 LABEL INSTR
60010 IF I=0 THEN LET I=1
60020 C$=LEFT$(B$,1)
60030 IF I<=LEN(A$) THEN GOTO 60050
60040 GOTO 60110
60050 FOR P=I TO LEN(A$)
60060 IF C$=MID$(A$,P,1) THEN GOTO 60090
60070 NEXT P
60080 GOTO 60110
60090 L=LEN(B$)
60100 IF B$=MID$(A$,P,L) THEN RETURN
60110 P=0
60120 RETURN
```

SPACE$(N)
STRING$(N, X$)

Application: SPACE$ retourne une chaîne composée de N blancs, et STRING$ une chaîne contenant N fois le premier caractère de X$.

Equivalent Lynx: Il suffit de déclarer en début de programme une chaîne, contenant un certain nombre de fois le caractère désiré (Par ex., S$="––––––––––––––––"). Ensuite, il suffira de faire LEFT$(S$, N) pour obtenir un équivalent de STRING$ ou SPACE$.

Exemples :

```
10 DIM S$(40)
20 S$="----------------------------------------"
100 ? LEFT$(S$,10);"RESULTATS";LEFT$(S$,10)
110 ? LEFT$(S$,29)

Donnera:
                    ----------RESULTATS----------
                    -----------------------------
```

```
100 CLS
110 DIM S$(20),T$(20)
120 S$="                    "
130 T$="--------------------"
140 PRINT@ 3,5;
150 FOR I=1 TO 20
160 PRINT LEFT$(T$,I)
170 NEXT I
180 PRINT
190 PRINT"POUR EFFACER, TAPEZ UNE TOUCHE"
200 X$=GET$
210 PRINT@ 3,5;
220 FOR I=1 TO 20
230 PRINT LEFT$(S$,I-1)
240 NEXT I
```

REMARQUE: On utilise aussi STRING$ pour dessiner rapidement des cadres de présentation, souligner du texte etc. SPACE$ est utile pour effacer une partie d'une ligne ou un mot à l'écran:

```
10 S$="
15 SPEED 0
20 CLS
30 FOR I=1 TO 400
40 ? CHR$(RAND(96)+32);
50 NEXT I
60 SPEED 70
70 PRINT@ (RAND(121)+3),(RAND(12)*10)+5;LEFT$(S$,RAND(6));
80 GOTO 70
```

2

La mémoire du Lynx

2.1. *Introduction*

Un certain nombre d'instructions et fonctions de Basic Lynx réalisent des accès directs à la mémoire. La suite de ce livre se réfère fréquemment à ces commandes, dans le chapitre 3 pour manipuler l'écran ou générer des caractères, et dans le chapitre 4 pour traiter le graphique avec des méthodes originales. Il apparaît donc comme indispensable, avant de présenter cette nouvelle section du Basic, de préciser quelques points sur la manière dont le Lynx manipule des informations et sur son organisation interne.

Une fois les instructions expliquées, ce deuxième chapitre se termine par une présentation du programme Moniteur et de ses fonctions spécifiques, et par une liste de quelques adresses utiles.

2.2. Le stockage des données en mémoire

Rappelons les quelques principes de base qui sont valables pour tout ordinateur 8 bits, et donc pour le Lynx:

— L'unité d'information la plus élémentaire est le **Bit**, qui ne peut prendre que deux valeurs, Ø ou 1. Pour les manipuler plus facilement, ceux-ci ne sont pas traités individuellement mais regroupés en "mots" de 8 bits, ou **Octets**.

— Un octet peut prendre 256 valeurs différentes (2^8), de Ø à 255. Il constitue l'unité de stockage pour toutes les informations transitant par la mémoire, ainsi que la plus petite unité accessible à l'utilisateur. Certaines données, comme le code d'un caractère, peuvent se stocker sur un seul octet; pour des valeurs plus grandes que 255, deux octets sont nécessaires. Chacun des 8 bits constituant l'octet peut prendre une valeur obtenue par sa valeur propre (Ø ou 1), multipliée par son **poids** qui dépend de sa position à l'intérieur de l'octet:

	←			OCTET			→	
Rang →	7	6	5	4	3	2	1	Ø
Poids →	2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1

— Le poids de chaque bit se calcule par puissances de deux croissantes. Le bit de droite de rang Ø est appelé "le moins signifiant", et celui de gauche (de rang 7), "le plus signifiant". Pour obtenir donc la valeur d'un octet, on multiplie le poids de chacun des bits le constituant par sa valeur (Ø ou 1), et on additionne le tout: Si, par exemple, les Bits de "poids" 16, 8 et 1 sont à 1 et tous les autres à Ø, l'octet aura la valeur 16+8+1=25. On voit que grâce à cette méthode de codage, un octet peut donc prendre toutes les valeurs entières comprises entre Ø (tous les bits à Ø) et 255 (tous les bits à 1).

— Pour savoir où est rangée chaque information et y accéder rapidement, l'ordinateur utilise des **Adresses**: Chaque octet est accessible par son adresse, qui permet de le définir par rapport à ses voisins. Les valeurs

des adresses étant comprises entre Ø et 65535 sur tous les Basic, deux octets sont nécessaires si l'on veut les stocker, car un octet ne peut pas prendre une valeur supérieure à 255. Dans le principe de codage, toute adresse est divisée en deux octets :

- Le premier, appelé *octet de poids faible* est le reste de la division entière de l'adresse par 256 (<adresse> MOD 256).
- Le deuxième, appelé *octet de poids fort*, est le résultat de cette division entière (<adresse> DIV 256).

— Pour relire une adresse codée sur deux octets, on ajoute la valeur de l'octet de poids faible à celle de l'octet de poids fort multipliée par 256 : <Adresse>=<octet 1>+<octet 2>*256.

— Pour une valeur supérieure à 255 codée sur deux octets, l'octet de poids faible est toujours contenu dans l'adresse la plus basse.

— Deux instructions permettent d'écrire directement dans une ou deux adresses, POKE et DPOKE. Les fonctions correspondantes, prévues pour lire le contenu de une ou deux adresses, sont PEEK et DPEEK. La syntaxe régissant l'emploi de ces commandes est détaillée au paragraphe 2.4.2.

2.3. *L'organisation mémoire*

La mémoire d'un ordinateur peut être comparée à un livre, divisé en chapitres d'inégale importance, et traitant de sujets différents. Chacun d'eux n'est susceptible de contenir qu'un type précis d'information. On peut énumérer :

— La ROM ou mémoire morte, qui représente le "sommaire du livre", contient le langage Basic du Lynx et ne peut être modifiée.

— La RAM, ou mémoire vive. Cette zone est entièrement accessible à l'utilisateur pour écrire des programmes, mais contient également les informations concernant le contenu de l'écran et les variables en cours d'utilisation. L'organisation interne du Lynx est assez originale, et peut être schématisée dans un plan que l'on appelle **Memory Map** :

MEMORY MAP – LYNX 96 K.

PAGE Ø	PAGE 1	PAGE 2	PAGE 3	ADRESSES
//////////	MEMOIRE VIVE 64 K	//////////	//////////	65535
//////////		BANQUE ROUGE 8 K	BANQUE VERT 8 K	57344
//////////		//////////	BANQUE VERT ALT. 8 K	49152
//////////		BANQUE BLEU 8 K	//////////	4Ø96Ø
//////////		//////////	//////////	32768
R.O.M. 24 K.	●●●●●●●●●●●	//////////	//////////	24576
	●●●●●●●●●●●	//////////	//////////	16384
	●●●●●●●●●●●	//////////	//////////	8192
				Ø

Légende : //////// Zones fictives de la mémoire, non disponibles.

●●●●●●● Parties de la RAM ayant les mêmes adresses que la ROM, inaccessible au Basic.

Explications :

Sur le Lynx 96 K, quatre pages mémoire sont disponibles, numérotées de Ø à 3. Pour chacune de ces zones, l'adressage va de Ø à 65535.

Page Ø Cette zone contient les 24 K de ROM du Lynx, entre les adresses Ø et 24574. Les autres adresses sont inutilisées.

Page 1 Ce sont les 64 K de mémoire vive utilisateur, de la version 96 K (la numérotation va de Ø à 65535). La zone représentée "●●●●" correspond aux 24 K de RAM ayant les mêmes adresses que la ROM. Cette zone ne peut être utilisée par l'intermédiaire du Basic. On dispose donc approximativement de 4Ø Kiloctets pour programmer sur le Lynx 96 K.

Dans le cas de la version 48 K, la zone de mémoire vive utilisateur est comprise entre les adresses 24576 et 4Ø96Ø. Sur cette version, c'est donc de 16 K dont on dispose pour les programmes.

Page 2 Cette portion de 16 K de mémoire vive, à laquelle on ne peut accéder directement, est située entre les adresses 32768 et 57344 et destinée à stocker le contenu de deux banques de couleur, bleu et rouge. Ces banques sont utilisées pour toutes les fonctions d'affichage et peuvent être déconnectées. Vous trouverez l'explication complète du mode d'affichage au paragraphe 3.1.2.

Page 3 Zone de nature identique à la précédente, mais contenant les banques de couleurs vert et vert alternatif. Cette dernière banque ne semble pas être utilisée en fonctionnement normal ; on peut toutefois s'en servir pour créer une deuxième page écran, méthode exposée au paragraphe 3.1.2.

REMARQUES :

— L'accès direct à la page 1 est possible par PEEK et POKE. Les pages 2 et 3 peuvent être déplacées en bloc pour modifier le fonctionnement de l'affichage, mais on ne peut y écrire.

— Les 64 K de mémoire utilisateur et les deux pages de 16 K qui correspondent aux banques de couleur représentent les 96 K de mémoire du Lynx 96 K. Sur le 48 K, seule la quantité de mémoire vive de la page 1 est diminuée (16 K), les banques de couleur restant identiques.

— La zone de mémoire vive de la page 1, comprise entre la dernière adresse de la ROM (24576) et l'adresse 26957, contient toutes les

variables-système du Lynx. Ces variables sont des pointeurs codés sur deux octets, contenant les adresses importantes du système (cf. § 2.6). Ces 2 Koctets de mémoire réduisent d'autant la place disponible pour le Basic (13.5 K. pour la version 48 K, et 38.5 K pour la version 96 K).

2.4. Commandes agissant sur la mémoire

2.4.1. Le principe des accès mémoire

Toutes les fonctions du Lynx passent par le processeur central Z 8Ø. Le Basic est un langage destiné à l'utilisateur, qui est interprété et converti en code-machine, le seul qui peut être compris par le Z 8Ø. Toutes les routines traduisant les instructions du Basic sont contenues dans la mémoire morte du Lynx, on ne peut donc les modifier. Trois types d'actions sont tout de même possibles pour le programmeur :

1. Effectuer des accès directs à la mémoire, c'est-à-dire lire et écrire directement dans des adresses. Pour ce faire, on utilise les instructions POKE et DPOKE, ainsi que les fonctions PEEK et DPEEK. De nombreuses manipulations sont possibles, à condition de bien connaître la structure-mémoire de l'ordinateur. De plus, certaines fonctions renvoient les adresses les plus importantes de la mémoire, pour permettre la modification de leur contenu. C'est le cas, entre autres, de HIMEM, HL, ALPHA, GRAPHIC. (Une liste des adresses qu'on ne peut obtenir par des fonctions spécialisées est donnée au § 2.6).

2. Écrire des programmes en langage-machine, c'est-à-dire des suites de codes ayant une signification pour le Lynx. Un programme de ce type s'exécute à partir du Basic avec CALL. Là encore, le Lynx offre des facilités avec les commandes CODE et LCTN. La présence d'un programme Moniteur en ROM apporte une grande souplesse dans ce domaine (§ 2.5).

3. Enfin, on peut agir sur les ports du Z 8Ø avec INP et OUT. Toute action à partir du clavier vers l'écran, ou à partir d'un quelconque périphérique vers l'unité centrale, passe par les ports d'entrée/sortie du Z 8Ø. En lisant et en envoyant des données dans ces ports, on peut modifier les

affichages-écran ou saisir des données au clavier, avec parfois plus de souplesse qu'en Basic (voir § 3.1.5 et § 4.1.5).

2.4.2. Instructions

CALL

Syntaxe : CALL <Adresse> [,A]

Application : Cette instruction sert à appeler une routine en langage-machine, préalablement écrite en mémoire et débutant à l'adresse donnée comme argument de CALL. Il est noter qu'on ne retournera au Basic que si un RTS (instruction assembleur équivalente à RETURN) termine la routine. Le passage d'un paramètre à la routine est possible, en transférant la valeur de A dans le registre HL du Z 8Ø. A est une variable ou une constante comprise entre Ø et 65535, qui sera arrondie à l'entier inférieur si elle comporte des décimales. Des choses inattendues se passeront si vous appelez des routines de la ROM, mais l'ordinateur se bloquera souvent.

Exemple : CALL Ø

Équivaut à un RESET par programme, avec le logo de mise en route en prime.

CODE

Syntaxe : CODE <octet 1> <octet 2>...<octet n>

Application : CODE permet l'implantation d'une routine en langage-machine en mémoire, à partir d'un programme Basic. Les octets, constituant les divers pas de la routine, sont des valeurs hexadécimales séparées par un espace. La ligne contenant CODE est ignorée par le programme Basic, seule l'instruction LCTN permet de connaître l'emplacement mémoire de ces octets.

REMARQUE : Les octets sont entrés directement en hexadécimal, sans les faire précéder d'un "&" (la notation décimale est donc impossible).

Exemple :

```
1Ø CODE C2 A9 B7
2Ø A=LCTN(1Ø)
3Ø CALL A
```

DPOKE

Syntaxe : DPOKE A,X

Application : Cette instruction est utile pour charger deux adresses consécutives de la mémoire (A et A+1) avec un nombre X codé sur deux octets. L'adresse A reçoit l'octet de poids faible et A+1 l'octet de poids fort. A et X sont des variables ou constantes comprises entre Ø et 65535, et seront arrondies à l'entier inférieur.

REMARQUE : DPOKE A,X est l'équivalent exact de :

```
POKE A,(X MOD 256)
POKE A+1,(X DIV 256)
```

Exemple :

```
DPOKE 16ØØØ,29Ø
```

Stocke à l'adresse 16ØØØ l'octet 34 et à l'adresse 16ØØ1 l'octet 1.

EXT

Syntaxe : EXT <Nom d'instruction>

Application : EXT est employé pour ajouter de nouvelles instructions au Basic du Lynx. Ces instructions doivent être entrées en code machine et placées à des adresses spécifiques.

Exemple :

```
EXT SPRINT/PPRINT
```

Sont deux instructions implantées sur la version 96 K pour la gestion des imprimantes.

LCTN

Syntaxe : LCTN (<N° de ligne>)

Application : On utilise LCTN pour connaître l'implantation en mémoire des codes inclus dans une ligne de programme Basic contenant l'instruction CODE. LCTN cherche la ligne spécifiée, et renvoie l'adresse du premier octet qui suit CODE. On peut ensuite exécuter la routine avec CALL <Adresse trouvée>.

MON

Syntaxe : MON

Application : Appelle le programme Moniteur, avec sortie du Basic (voir § 2.5).

REMARQUES :

— Le programme Basic et ses données ne sont pas effacés de la mémoire.

— MON affiche le contenu des registres du Z 8Ø, dans l'ordre suivant : AF, HL, DE, BC, AF', HL', DE', BC', IX, IY, SP, PC.

— Il faut frapper la touche "J" pour revenir au Basic.

OUT

Syntaxe : OUT P,<octet>

Application : OUT envoie un octet sur le port P du Z 8Ø. <octet> doit être compris entre Ø et 255.

REMARQUE : L'emploi des ports, bien qu'assez complexe, permet des "trucs" apportant des compléments au Basic (voir § 3.1.5, 4.1.5).

POKE

Syntaxe : POKE A,X

Application : Modifie le contenu de l'adresse A, en y plaçant l'octet X. A est un entier compris entre Ø et 65535, X lui est compris entre Ø et 255.

REMARQUE : La fonction PEEK est le complément de POKE. Une utilisation erronée de POKE peut bloquer le fonctionnement de l'ordinateur.

Exemple :

```
POKE 26957,192
```

Réintègre en mémoire un programme détruit par NEW.

RESERVE

Syntaxe : RESERVE <adresse>

Application : Fixe l'adresse de la plus haute position mémoire accessible

par le Basic, appelée HIMEM. La portion de mémoire au-dessus de HIMEM est alors protégée, et peut servir pour stocker des données ou des programmes en langage-machine.

REMARQUE: Si <adresse> est supérieure à HIMEM, une erreur est provoquée.

2.4.3. Fonctions

ALPHA

Syntaxe: ALPHA

Application: ALPHA est un pointeur, c'est-à-dire une constante contenant la première adresse d'un couple servant lui-même à coder une adresse importante de la mémoire. Dans le cas d'ALPHA, l'adresse pointée est celle contenant l'emplacement du jeu de caractères alphanumérique standard (caractères dont les codes sont compris entre 32 et 127).

REMARQUE: Le contenu de l'adresse pointée par ALPHA peut être modifié, afin de déplacer la position où le jeu est lu en mémoire.

Exemples:

PRINT ALPHA retourne 25199 sur le Lynx 96 K.

PRINT DPEEK(ALPHA) retourne 148, c'est le point où démarre le jeu de caractères standard du Lynx.

DPOKE ALPHA, GRAPHIC fait démarrer la lecture du jeu de caractères standard au début de la table des caractères semi-graphiques (voir GRAPHIC). L'action sur les touches retourne des mosaïques à la place du jeu ASCII.

DPEEK

Syntaxe: DPEEK (A)

Application: Cette fonction nous permet de connaître le contenu de deux adresses mémoire consécutives (A et A+1). Il s'agit bien sûr du complément de l'instruction DPOKE. L'adresse A contient l'octet de poids faible et A+1, celui de poids fort.

REMARQUE: PRINT DPEEK(A) est équivalent à
PRINT PEEK(A)+(PEEK(A+1)*256)

Exemple :

PRINT DPEEK(25181)

Renvoie la vitesse de clignotement du curseur, modifiable par CFR.

GRAPHIC

Syntaxe : GRAPHIC

Application : Comme ALPHA, il s'agit d'un pointeur, mais renvoyant l'adresse contenant le point de démarrage en mémoire du jeu de caractère alternatif, dont les codes sont compris entre 128 et 249. Ce jeu comprend une copie du jeu normal ASCII entre les codes 128 et 223, et la table des mosaïques graphiques entre 224 et 249.

REMARQUE : GRAPHIC est employé pour déplacer le jeu alternatif en mémoire lors de la création de caractères, traitée au paragraphe 3.2.2.

Exemple :

? GRAPHIC Retourne 25201

? DPEEK (GRAPHIC)

Retourne 468, l'adresse de début en mémoire du jeu alternatif.

HIMEM

Syntaxe : HIMEM

Application : Retourne la plus haute position mémoire accessible par le Basic.

REMARQUE : On peut modifier HIMEM grâce à l'instruction RESERVE.

Exemples : RESERVE 49151 place HIMEM en 29151.
RESERVE HIMEM–30 décale HIMEM de 30 octets.

HL

Syntaxe : HL

Application : Renvoie le contenu du registre HL, après exécution d'un CALL avec passage de variable.

REMARQUE : Si un CALL n'a pas été exécuté, HL retourne 0.

INP

Syntaxe : INP (P)

Application : Retourne la valeur de l'octet présent au port P du Z 8Ø.

REMARQUE : INP est la fonction inverse de OUT.

LETTER

Syntaxe : LETTER (C)

Application : Calcule la première adresse en mémoire du caractère de code C.

REMARQUE : On utilise surtout LETTER pour la modification de caractères, dont les codes sont compris entre 128 et 249 (voir § 3.2.2).

Exemple :

```
? LETTER(249)
```

Renvoie 1678, c'est la première des huit adresses-mémoire du pavé en vidéo inverse de code 249.

MEM

Syntaxe : MEM

Application : Retourne l'espace mémoire restant disponible pour le Basic.

Exemple :

```
1ØØØØ LABEL MEM
1ØØ1Ø IF MEM < 2ØØ THEN ?"PLUS DE MEMOIRE"
1ØØ2Ø RETURN
```

Ce sous-programme, appelé régulièrement dans un programme "gourmand" en mémoire, avisera l'utilisateur en cas de pénurie !

PEEK

Syntaxe : PEEK (A)

Application : Donne la valeur de l'octet contenu dans l'adresse A. PEEK est le complément de POKE.

Exemple :

? PEEK(25141)

Renvoie le dernier caractère entré au clavier.

2.5. Le programme moniteur

Le Moniteur est un logiciel implanté dans la mémoire morte du Lynx qui supervise toutes les fonctions du Basic, et contient les utilitaires de base du système. Ces utilitaires en langage-machine sont habituellement exploités par le Basic, et à priori ne concernent pas l'utilisateur. En fait, ces routines sont très facilement accessibles à l'aide de MON, qui passe la main au programme Moniteur.

Une liste d'instructions est alors disponible, sous la forme de commandes comprenant une seule lettre. Le seul inconvénient réside dans l'impossibilité d'inclure ces puissantes instructions à l'intérieur d'un programme Basic. Nous verrons au paragraphe 2.5.2 comment tourner cette difficulté.

REMARQUES :

— Toute commande du Moniteur est entrée sous la forme :
  <Commande> X Y... suivis de RETURN
  où X et Y sont des nombres hexadécimaux séparés d'un espace.
— Lors de l'emploi du Moniteur, le curseur est fixe et s'affiche sous la forme d'un astérisque.
— Une erreur d'entrée provoque l'apparition de quatre points d'interrogation (????).

2.5.1. Les commandes du moniteur

ARITHMETIC

Syntaxe : **A** X Y

Application : La commande A calcule et affiche X+Y, X–Y, Z. Z représente

le saut relatif à effectuer pour aller de X à Y (nombres d'adresses les séparant). C'est la seule instruction de calcul du Moniteur.

Exemple :

A AF AØ renvoie Ø14F ØØØF EF

BREAKPOINT

Syntaxe : **B** [X]

Application : Interrompt l'exécution d'un programme en langage-machine, lorsque l'adresse X est atteinte. BREAKPOINT passe la main au Moniteur, qui autorise l'examen des registres et leur modification. B sans argument redémarre l'exécution.

REMARQUE : B est équivalent à STOP et CONT en Basic.

COPY

Syntaxe : **C** X Y Z

Application : Copie un bloc de Z octets de mémoire, de la zone débutant à l'adresse X vers la zone débutant à l'adresse Y.

REMARQUES :

— Si les zones se chevauchent, celle démarrant à X est modifiée au fur et à mesure de la copie. (Voir INTELLIGENT COPY.)

— COPY est utile pour déplacer une zone mémoire, et s'exécute très rapidement.

Exemple :

C AØØØ BFFF FF

Copie 255 (FF) octets de l'adresse 4Ø96Ø (AØØØ) à l'adresse 49151 (BFFF).

C AØØØ AØØ1 FF

Les deux zones se chevauchant, le contenu de l'adresse AØØØ est recopié successivement sur 255 octets.

DUMP CASSETTE

Syntaxe: **D** X Y Z "<Nom>"

Application: Copie sur le K7 la zone mémoire comprise entre les adresses X et Y. Le nom permet de recharger ensuite la zone mémoire en l'associant à un identificateur. L'argument Z spécifie l'adresse à laquelle le programme démarre l'exécution après son rechargement. Si la zone mémoire sauvegardée ne correspond pas à un programme exécutable, mais à un bloc de données, il faut entrer Z=Ø.

REMARQUES:

— DUMP est la seule fonction du Lynx permettant la sauvegarde sur K7 de données ou de variables. On l'utilise dans le chapitre 3, pour conserver les jeux de caractères créés avec le programme GENECAR, et dans le chapitre 5 afin de sauvegarder des mélodies créées par le programme CHOPIN. La méthode est toujours la même: On range les données à sauvegarder dans un tableau, on copie le contenu de ce tableau dans la zone mémoire démarrant à HIMEM (après l'avoir abaissée du nombre d'adresses désiré avec RESERVE), et on transfère cette zone sur K7 à l'aide de DUMP.

— Une fois les données sauvegardées, la commande R permet de les recharger par l'intermédiaire du Moniteur, et l'instruction MLOAD à partir du Basic (voir § 1.5.4).

— La zone est rechargée à l'endroit qu'elle occupait lors de sa sauvegarde.

Exemple:

```
D 6256 6259 Ø "Window"
```

Sauvegarde sur K7 les quatre paramètres de l'instruction WINDOW. L'exécution de MLOAD "Window" réinitialise la fenêtre-écran avec ces paramètres (§ 3.1.2).

EXECUTE

Syntaxe: **E** [X]

Application: Cette commande réinitialise le contenu des registres du Z 8Ø, et exécute le programme stocké à l'adresse X. E sans argument démarre l'exécution à partir de l'adresse stockée dans le registre PC (Compteur Programme). On peut l'utiliser comme B (BREAKPOINT).

REMARQUE: E est équivalent au RUN Basic.

FILL

Syntaxe: **F** X Y Z

Application: Place l'octet Z dans toutes les adresses mémoire comprises entre X et Y inclus.

GO

Syntaxe: **G** X [Y]

Application: Exécute le programme débutant à l'adresse X, sans modification des registres du Z 8Ø. Si le paramètre Y est ajouté, sa valeur est chargée dans le registre DE. On peut donc par intermédiaire passer une variable au programme.

REMARQUE: G est l'équivalent de CALL en Basic.

DUMP HEXADECIMAL

Syntaxe: **H** [X]

Application: Affiche à l'écran le contenu des 128 adresses mémoire suivant X. Chaque ligne est imprimée sous le format suivant:

— La première adresse d'un groupe de huit.
— Les huit octets suivant cette adresse.
— Les huit caractères ASCII correspondant aux octets. Si aucun caractère ne correspond au code, un point "." est affiché.

Si X est omis, sa valeur par défaut correspond à la dernière adresse pointée par une instruction du Moniteur.

REMARQUE: Une fois les 16 lignes d'octets affichées, il suffit de rentrer H sans argument pour visualiser les 128 suivants.

INTELLIGENT COPY

Syntaxe: **I** X Y Z

Application: Identique à COPY (C), en effectuant celle-ci par décalage de 2 octets à la fois seulement afin d'éviter les problèmes posés par les zones se chevauchant.

Exemple :

C AØØØ AØØ1 FF

Décale d'une adresse la zone de 255 octets démarrant à AØØØ.

JUMP

Syntaxe : **J**

Application : Abandon du Moniteur et retour au Basic.

LOCATE

Syntaxe : **L** X Y

Application : Recherche la présence de l'octet Y dans toute la mémoire à partir de l'adresse X. Chaque fois que l'octet est rencontré, son adresse est affichée.

REMARQUE : ESC Interrompt ce mode.

MODIFY

Syntaxe : **M** [X Y_1 Y_2 ... Y_n]

Applications : Autorise l'examen d'une zone mémoire adresse par adresse, ainsi que sa modification. Deux méthodes sont disponibles :

1. M X, où X représente l'adresse à modifier, affichera celle-ci suivi de son contenu entre crochets. Un curseur clignotant indique que l'on peut entrer un octet qui remplacera le contenu de X (en cas d'omission de X, la dernière adresse pointée par le moniteur sera prise par défaut) :

- Si un seul octet suivi de <RETURN> est entré, M passe, dans le même mode, à l'adresse suivante.

- Si par contre on entre une suite d'octets séparés d'espaces, ils sont rangés à partir de l'adresse X dans l'ordre de leur entrée. M passe ensuite à l'adresse suivant celle du dernier octet modifié.

- Si on entre uniquement <RETURN>, M passe à l'adresse suivante sans modifier le contenu de X. Cette fonction est pratique pour examiner un bloc d'adresses pas à pas.

2. M suivi de X et d'une succession d'octets, range ceux-ci dans l'ordre de leur entrée à partir de l'adresse X, puis affiche l'adresse suivant celle du dernier octet modifié.

REMARQUES :

— Un "/" suivi de <RETURN> sur le curseur clignotant provoquera le retour à l'adresse précédente.

— Un "." suivi de <RETURN> permet de sortir du mode MODIFY.

OUTPUT

Syntaxe : **O** X Y

Application : OUTPUT envoie l'octet Y sur le port X d Z 8Ø.

REMARQUE : Équivalent à l'instruction OUT du Basic.

INCREMENT

Syntaxe : **P** X

Application : Incrémente le contenu du registre PC (Compteur Programme) de la valeur X.

QUERY PORT

Syntaxe : **Q** X

Application : Lit et affiche l'octet présent sur le port X du Z 8Ø.

REMARQUE : QUERY correspond à INP en Basic.

READ CASSETTE

Syntaxe : **R** "<Nom>"

Application : Relit sur K7 le programme ou la zone mémoire nommée.

REMARQUE : READ correspond à MLOAD en Basic.

SCREEN CLEAR

Syntaxe : **S**

Application : Efface l'écran sous Moniteur.

TYPE

Syntaxe: **T** X <suite de caractères>

Application: Stocke le code ASCII des caractères entrés, à partir de l'adresse X.

Exemple :

```
T AØØØ COMPUTER   RETURN
H AØØØ            RETURN
```

UPDATE

Syntaxe: **U** <Nom du registre> X

Application: Remplace le contenu du registre spécifié par X, puis affiche le contenu de tous les registres dans le même ordre que lors du passage en Moniteur.

Exemple :

```
X                 RETURN
U AF Ø            RETURN
```

VERIFY

Syntaxe: **V** X Y Z

Application: Vérifie que la zone mémoire démarrant à l'adresse X et de longueur Z octets, est identique à celle débutant à l'adresse Y. Lors de la recherche, les adresses dont le contenu est différent sont affichées.

WORD

Syntaxe: **W** X Y

Application: W est identique à LOCATE, mais cherche un mot codé sur deux adresses (Y), à partir de l'adresse X.

REMARQUE: Dans les couples d'adresses examinés, la plus basse est sensée contenir l'octet de poids le plus faible.

REGISTER 1

Syntaxe : **X**

Application : Affiche le contenu des registres du Z 8Ø, dans le même ordre qu'à l'appel du Moniteur :

AF HL DE BC
AF' HL' DE' BC' IX IY SP PC

REGISTER 2

Syntaxe : **Z**

Application : Liste en colonnes les registres du Z 8Ø, suivis de leur contenu et des huit octets qu'ils pointent.

2.5.2. L'emploi du moniteur sous Basic

L'intérêt du Moniteur réside sans conteste dans les fonctions auxquelles il permet d'accéder et qui sont irréalisables par le Basic. L'inclusion de fonctions du Moniteur dans un programme est possible par une méthode simple : Il suffit d'écrire à l'écran, juste avant la sortie du Basic (MON), un commentaire contenant des instructions expliquant la marche à suivre sous Moniteur, ainsi que le moyen de retourner en Basic.

Exemple : Affichage des registres du Z 8Ø.

```
100 REM REGISTRES
110 WINDOW 3,123,5,245
120 CLS
130 VDU 18
140 PRINT@ 3,205;"POUR AFFICHER LES REGISTRES, TAPER Z..
160 PRINT@ 3,215;"POUR RETOURNER DANS LE PROGRAMME,.TAPE
SUIVI DE <RETURN> PUIS GOTO LABEL GO.."
180 PRINT@ 3,5;"AFFICHAGE DES REGISTRES DU Z80.........'
190 VDU 18
200 WINDOW 3,123,15,195
210 VDU 23
220 MON
230 LABEL GO
240 WINDOW 3,123,5,245
250 CLS
260 PRINT"RETOUR AU BASIC EFFECTUE. TERMINE"
```

REMARQUE: Les points à la fin des commentaires servent à compter les espaces.

Explications:

110,120: Réinitialise l'écran.

130 : Sélection de la vidéo inverse.

140,180: Affichage des commentaires et explications.

190 : Retour en vidéo normale.

200 : Sélection de la fenêtre-écran où seront affichés les registres.

210 : Le curseur est placé dans cette fenêtre (pour tous les codes VDU, voir § 3.1.1).

220 : Appel du Moniteur.

230,260: Fin du programme.

2.6. Quelques adresses utiles

Cette liste d'adresses apporte des informations très utiles sur l'organisation du Basic et des variables. Elle est incomplète, car les adresses concernant la gestion d'écran, le graphique et la couleur sont traitées dans les chapitres correspondants.

Rappel: Ces adresses sont des **pointeurs**, c'est la lecture de leur contenu qui retourne le point recherché, et non leur valeur elle-même.

2.6.1. Les pointeurs-système

DPEEK (25082): Adresse de début de la zone programme en mémoire.

26957 : Adresse du premier octet de la première ligne de programme.

DPEEK (25084): Fin de la zone programme, début des variables numériques.

DPEEK (25119) : Sommet des variables.

REMARQUES :

— Les variables-chaîne sont stockées à partir de leur sommet vers l'adresse la plus basse du Basic, tandis que programme et variables numériques le sont à partir de la dernière adresse de la ROM vers le sommet de la mémoire. Une zone tampon sépare sommet des chaînes et sommet des variables numériques, évitant le chevauchement de ces deux sections.

— On peut, par l'intermédiaire du Moniteur (DUMP), stocker ces zones sur cassette pour la sauvegarde de variables dans un programme.

2.6.2. Les adresses fonctionnelles

DPEEK (25111) : Adresse de démarrage en mémoire de la table des noms de fonctions acceptés par le Basic Lynx. On peut examiner cette table par l'intermédiaire du Moniteur à l'aide de la commande H, ou en Basic avec le programme suivant :

```
100 CLS
105 A=DPEEK(25111)
110 FOR I=A TO A+361
120 A$=CHR$(PEEK(I))
130 IF ASC(A$)>96 THEN LET A$=CHR$(ASC(A$)-32)
140 IF PEEK(I)>32 THEN ? A$;
150 NEXT I
```

Explications :

110 : Boucle parcourant la table à partir de l'adresse de début.

120 : Saisie d'un caractère de la table.

130 : Conversion en majuscules.

140 : Impression des caractères.

DPEEK (25113) : Cette adresse pointe sur la table des Instructions du Basic. On peut l'examiner avec le même programme, en modifiant la ligne 105.

DPEEK (25Ø97) : Après interruption par STOP ou ESC, ce pointeur contient l'adresse de redémarrage du programme utilisée par l'instruction CONT. Si l'exécution ne peut être reprise, DPEEK (25Ø97) renvoie Ø. On peut forcer le point de démarrage en donnant à ce pointeur la valeur de l'adresse de début d'une autre ligne de programme : DPOKE 25Ø97,A.

DPEEK (25139) : Stocke la valeur déterminant le temps mis par le Lynx pour déclencher l'autorépétition des touches du clavier (2Ø48 à la mise sous tension). Pour diminuer ou allonger ce temps, il suffit de modifier cette valeur par DPOKE. Une valeur très élevée supprime l'autorepeat.

PEEK (25145) : Stocke sur un octet l'état de <SHIFT LOCK>. Ø correspond au blocage majuscules, et 1 au mode normal. On peut donc forcer par programme l'état du clavier, quel que soit son état précédent (alors que la touche <SHIFT LOCK> fonctionne en bascule).

Exemples :

```
10 CLS
20 POKE 25145,0
30 X$=GET$
40 ? X$;
50 GOTO 20
```

Le clavier reste en majuscule, sauf en cas d'appui continu sur SHIFT.

PEEK (25141) : Contient le code du dernier caractère entré au clavier.

PEEK (25146)
à (25171) : Ces 26 adresses correspondent aux 26 codes des instructions générées par le One-Key Basic (ESC A à Z). On peut, par leur intermédiaire, modifier la configuration clavier en déplaçant ou modifiant les codes. Rappelons que toute instruction ou fonction possède un code sur un octet.

Exemple : POKE 25146,47 remplace AUTO par LABEL en frappant ESC A.

Voici la liste complète des codes en One-Key Basic :

Touche	*Instruction*	*Code*	*Adresse*
A	AUTO	28	25146
B	BEEP	48	25147
C	CONT	21	25148
D	DEL	27	25149
E	DATA	65	25150
F	DEFPROC	2	25151
G	GOTO	16	25152
H	GOSUB	17	25153
I	INPUT	37	25154
J	LABEL	47	25155
K	MON	62	25156
L	LIST	40	25157
M	RETURN	18	25158
N	NEXT	5	25159
O	ENDPROC	4	25160
P	PROC	3	25161
Q	REM	23	25162
R	REPEAT	56	25163
S	STOP	14	25164
T	TRACE	68	25165
U	UNTIL	57	25166
V	VERIFY	9	25167
W	WHILE	54	25168
X	WEND	55	25169
Y	RUN	20	25170
Z	RESTORE	23	25171

Tous les codes des instructions et fonctions étant compris entre 1 et 255, il est facile de faire une liste complète de toutes les commandes en assignant à la même touche du clavier tous les codes successivement. Cette liste vous permettra ensuite d'effectuer les combinaisons les mieux adaptées à vos besoins.

On peut sauvegarder une configuration clavier sur K7, et la recharger par le Basic :

Sauvegarde :

MON	RETURN
D 623A 6254 "TABLE"	RETURN
J	RETURN

Chargement :

MLOAD "TABLE"	RETURN

3

La gestion de l'écran et les jeux de caractères

Cette section est consacrée à l'écran en mode texte, ainsi qu'à tous les codes, commandes et fonctions correspondantes. L'écran haute-résolution graphique est traité au chapitre 4.

3.1. La gestion de l'écran

L'écran est prévu pour afficher 24 lignes de 4Ø caractères chacune. Il est possible d'y mélanger texte et graphique haute-résolution, ainsi que d'y définir la taille d'une fenêtre d'écriture. Certains codes, appelés **Code de Contrôle**, autorisent une gestion très simple de la couleur et du positionnement du curseur. Enfin, un éditeur assez performant est prévu pour la mise au point des programmes.

Toutes ces fonctions, associées à l'emploi de quelques astuces en langage machine, apportent au Lynx une grande souplesse d'affichage et améliore la présentation des programmes.

3.1.1. Les codes de contrôle

L'affichage de tous les caractères du code ASCII est possible avec PRINT CHR$(C), ou C représente le code du caractère. Une instruction du Basic est prévue pour remplacer "PRINT CHR$(C);":

VDU

Syntaxe: VDU <code 1>,<code 2>,...<code n>

Application: Affiche à l'écran à la position du curseur les caractères correspondant à la liste de codes suivant l'instruction.

REMARQUE: Les caractères sont affichés côte à côte sans séparateurs.

Exemple:

```
VDU 65,87
```

Cette commande est surtout utile pour l'affichage de **Codes de Contrôle**. On appelle codes de contrôle les codes ASCII inférieurs à 32. Chacun d'eux correspond, non pas à un caractère, mais à une fonction du Basic concernant en général l'écran. Sur le Lynx, ces codes ne sont pas accessibles en mode direct par une simple combinaison de touches. Vous trouverez ci-dessous la ligne complète de ces codes.

VDU 1,<N° Couleur> Équivalent de INK <Couleur>, cf. § 3.1.3.

VDU 2,<N° Couleur> Équivalent de PAPER <Couleur>, cf. § 3.1.3.

VDU 4 Équivalent de CLS.

VDU 5 Déplace le curseur en mode texte de un pixel vers le haut (un caractère occupe 1Ø pixels).

VDU 6 Déplace le curseur de un pixel vers le bas.

VDU 7 Envoie une courte tonalité dans le HP.

VDU 8 Déplace le curseur d'un espace vers la gauche en effaçant le caractère précédent et en ramenant le restant de la ligne.

VDU 9 Déplace le curseur vers la droite, à la position de tabulation suivante (une position toutes les dix colonnes).

VDU 1Ø Passe à la ligne sur une même colonne (line feed).

VDU 12 Déplace le curseur d'un espace vers la droite, sans effacement.

VDU 13 Place le curseur au début de la ligne suivante en effaçant celle-ci.

VDU 14 Le curseur reste toujours visible sous la forme d'un trait de soulignement pendant l'exécution d'un programme.

VDU 15 Supprime le mode précédent;

VDU 16 Place le curseur en haut et à gauche de l'écran.

VDU 18 Intervertit les couleurs de fond et de premier plan : C'est l'affichage en vidéo inverse.

VDU 19 Effectue un VDU 13 si le curseur n'est pas déjà au début d'une ligne.

VDU 2Ø Supprime le mode "surimpression" (VDU 21).

VDU 21 Sélectionne le mode "surimpression". On put dans ce cas écrire par dessus les caractères présents à l'écran, sans les effacer. Si deux points ont les mêmes coordonnées, la couleur résultante est la combinaison de la couleur de ces deux points.

Exemple :

```
100 REM CODE VDU 21
105 PROTECT 0
110 VDU 2,1
120 CLS
130 VDU 1,0
140 FOR I=1 TO 960
150 PRINT "A";
160 NEXT I
180 VDU 1,4
190 VDU 24,21
200 PRINT@ 3,60;"TEXTE EN SURIMPRESSION";
210 VDU 25
220 PROTECT 0
230 VDU 1,7
240 PRINT@ 3,235;"FIN";
```

Explication :

1Ø5,13Ø :Efface l'écran, sélectionne un fond bleu et une encre noire.

14Ø,16Ø : Remplissage de l'écran.

18Ø : La couleur de l'encre est passée en vert.

19Ø : Sélectionne la double taille et la surimpression.

2ØØ,24Ø : Affichage de texte en surimpression.

REMARQUE : PROTECT permet également la surimpression par l'intermédiaire des couleurs (voir § 3.1.3).

VDU 22 Déplace le curseur d'un espace vers la gauche, sans effacement.

VDU 23 Replace le curseur dans le coin supérieur gauche de la fenêtre définie par WINDOW. Équivalent à VDU 16 si la fenêtre occupe la totalité de l'écran.

VDU 24 Sélectionne la double taille (en hauteur) pour les caractères.

REMARQUE : Les coordonnées de PRINT sont altérées par ce mode. La position d'écriture verticale standard va de 5 à 115 par pas de 1Ø. Les coordonnées horizontales restent identiques.

VDU 25 Annule le mode double taille.

VDU 28 Déplace le curseur en mode texte de trois pixels vers le haut.

VDU 29 Déplace le curseur de trois pixels vers le bas.

Exemple :

```
10 CLS
20 FOR I=1 TO 10
30 PRINT"A";
40 VDU 29
50 NEXT I
```

VDU 3Ø Efface la ligne courante vers la droite, à partir de la position du curseur.

VDU 31 Place le curseur au début de la ligne suivante, sans effacer celle-ci.

REMARQUES GÉNÉRALES :

— Certains codes VDU sont équivalents à des instructions Basic. Leur intérêt réside dans la possibilité de placer sur une seule ligne de programme une suite de codes, réalisant plusieurs commandes enchaînées : VDU 2,Ø,1,7,4,24 sélectionne un écran en blanc sur noir, l'efface, et passe en mode double hauteur.

— Les codes 1Ø, 12 et 22 permettent la gestion des déplacements du curseur par programme dans trois directions, et correspondent aux codes générés par les trois flèches du clavier :

↓ 1Ø correspond à VDU 1Ø
← 22 correspond à VDU 22
→ 12 correspond à VDU 12

Le déplacement du curseur vers le haut ne correspond pas à un code VDU, mais une lecture du code de la touche ↑ renvoie la valeur 11. On peut donc la traiter par le Basic pour réaliser une fonction équivalente : Il suffit d'écrire VDU 28,28,28,5 chaque fois qu'un code 11 est lu au clavier.

Exemple :

Ce programme montre les différentes possibilités d'affichage offertes par VDU en noir et vert (mode TEXT) : Vidéo inverse, soulignement, surintensité.

```
100 TEXT
110 VDU 1,7,4
130 FOR I=1 TO 840
140 IF I=121 THEN VDU 18
150 IF I=281 THEN VDU 18
160 IF I=401 THEN VDU 14,21
170 IF I=560 THEN VDU 15,21
180 IF I=681 THEN PROTECT 0
190 PRINT"A";
200 NEXT I
210 PROTECT MAGENTA
220 VDU 1,4
230 END
```

Explications :

1ØØ,12Ø : Initialisation. VDU 1,7 est sans effet en mode TEXT (voir § 3.1.3).

13Ø,2ØØ : Boucle d'affichage, incluant :

140 : Vidéo inverse.

150 : Annule ce mode.

160 : Soulignement par affichage du curseur en mode surimpression.

170 : Annule ce mode.

180 : PROTECT 0 valide le VDU 1,7 de la ligne 110 (surintensité par utilisation d'une couleur plus claire).

210,220: Retour au mode TEXT, sans effacer l'écran.

3.1.2. L'exploitation de WINDOW

Comme nous l'avons vu au paragraphe 1.4.2, WINDOW sert à redéfinir complètement les dimensions de l'écran en mode texte, étant entendu qu'un affichage standard correspond à:
WINDOW 3,123,5,245.

Une fois WINDOW exécuté, les instructions PRINT agissent à l'intérieur de la fenêtre, et le scrolling intervient au bas de celle-ci. Il est à noter que la redéfinition de WINDOW n'affecte pas les données présentes à l'écran.

Exemple :

```
100 REM FENETRE
110 VDU 1,7,2,0
120 WINDOW 3,123,5,245
130 CLS
140 FOR I=1 TO 960
150 PRINT "A";
160 NEXT I
170 WINDOW 33,93,45,205
180 VDU 23
190 VDU 18
200 FOR I=1 TO 18
210 PRINT
220 NEXT I
230 VDU 23
240 PRINT"FENETRE TEXTE"
250 PRINT@ 33,65
260 FOR I=1 TO 10
270 PRINT"INDEPENDANTE !!"
280 NEXT I
290 END
```

Explications :

110 : Sélectionne les couleurs de l'affichage (voir § 3.1.3).

120 : Initialisation de WINDOW.

130,160 : Remplit l'écran avec le caractère "A".

170 : Création d'une fenêtre.

180,190 : Placement du curseur dans la fenêtre et sélection de la vidéo inverse.

200,220 : Nettoie le contenu de la fenêtre.

230 : Le curseur est replacé dans le coin supérieur gauche de la fenêtre.

240,280 : Impression d'un texte.

REMARQUES :

— L'instruction CLS efface tout l'écran. Pour effacer uniquement le contenu d'une fenêtre, il faut utiliser un sous-programme qu'on appelle avec PROC EFFACER (X) :

```
60000 DEFPROC EFFACER (X)
60010 VDU 23
60020 FOR N= 1 TO X
60030 VDU 30,31
60040 NEXT N ,
60050 ENDPROC
```

La variable X représente le nombre de lignes de la fenêtre, c'est-à-dire la différence entre le quatrième et le troisième paramètre de WINDOW que divise 10.

— Il est possible d'écrire à l'extérieur d'une zone définie par WINDOW en utilisant l'instruction PRINT @, VDU 23 replace ensuite le curseur à l'intérieur de la fenêtre.

— Les paramètres de WINDOW doivent être entrés tous les quatre à chaque utilisation de cette instruction. On peut éviter cette opération fastidieuse en écrivant directement dans les adresses-mémoire qui contiennent ces paramètres :

25174 : Bord gauche de l'écran.

25175 : Bord droit de l'écran.

25176 : Bord supérieur.

25177 : Bord inférieur.

On peut modifier une seule des adresses sans provoquer d'erreur : POKE 25174,42 déplace le bord droit de l'écran uniquement. De même, on peut lire ces adresses avec PEEK pour connaître les dimensions de la fenêtre :

```
10 FOR I= 25174 TO 25177
20 ?PEEK(I)
30 NEXT I
```

Exemple :

Sous-programme d'effacement de fenêtre s'adaptant automatiquement aux dimensions existantes :

```
60000 DEFPROC EFFACER
60010 VDU 23
60020 L=INT((PEEK(25177)-PEEK(25176))/10)
60030 FOR N= 1 TO L
60040 VDU 30,31
60050 NEXT N
60060 ENDPROC
```

3.1.3. La couleur du Lynx

Organisation mémoire

Le rangement des couleurs en mémoire suit des règles très particulières : Les trois couleurs de base, rouge, bleu et vert sont rangées dans trois "banques" de 8K., situées dans les pages-mémoire 2 et 3 (voir memory map, § 2.3). L'apparition d'une de ces trois couleurs à l'écran s'obtient en écrivant les données dans les banques correspondantes. C'est en mélangeant ces diverses couleurs entre elles que l'on accède aux huit teintes disponibles.

JAUNE = ROUGE+VERT
MAGENTA = BLEU+ROUGE
CYAN = VERT+BLEU
BLANC = ROUGE+VERT+BLEU
NOIR = AUCUNE COULEUR

Trois pointeurs contiennent l'adresse de début en mémoire des trois banques. Il s'agit de :

DPEEK (25230) : Page 2, couleur Bleu.

DPEEK (25232) : Page 2, couleur Rouge.

DPEEK (25234) : Page 3, couleur Verte.

Ces pointeurs peuvent servir à modifier l'emplacement d'une banque à l'intérieur de sa page-mémoire. Par exemple :

```
10 B=DPEEK(25230),R=DPEEK(25232)
20 DPOKE 25230,R
30 DPOKE 25232,B
```

Ce petit programme intervertit le bleu et le rouge pour les instructions du Basic Lynx (1 sélectionne le rouge et 2 le bleu).

Enfin, une banque supplémentaire de 8K., nommée "Vert alternatif" et démarrant à l'adresse 40960 sur la page 3, peut servir à créer plusieurs pages-écran en mode monochrome noir et vert. Cette méthode est décrite ci-après en même temps que l'instruction TEXT.

Instructions agissant sur la couleur

INK

Syntaxe : INK C
INK <Nom couleur>

Application : Sélectionne la couleur d'écriture, en mode texte ou graphique. Les huit couleurs peuvent apparaître simultanément à l'écran, et sont indépendantes du fond sur lequel elles s'affichent. C représente un numéro de couleur, compris entre 0 et 7. On peut également associer INK au nom des couleurs en Anglais, sans l'inclure entre guillemets :

Numéro	*Nom*	*Couleur*	*Numéro*	*Nom*	*Couleur*
0	BLACK	NOIR	4	GREEN	VERT
1	BLUE	BLEU	5	CYAN	BLEU CIEL
2	RED	ROUGE	6	YELLOW	JAUNE
3	MAGENTA	ROSE	7	WHITE	BLANC

REMARQUES :

— INK est à la fois une instruction et une fonction. PRINT INK retourne le code de la couleur valide.

— INK ne modifie pas les couleurs déjà affichées, la couleur ne devient active que pour les affichages suivants.

— INK est équivalent à VDU 1.

PAPER

Syntaxe : PAPER C
PAPER <Couleur>

Application : PAPER spécifie la couleur du fond de l'écran. Son emploi suit les mêmes règles que INK.

REMARQUES :

— Pour changer la couleur de tout l'écran, il faut faire suivre PAPER de CLS.

— On peut associer PAPER à PRINT @ pour modifier la couleur d'une seule ligne d'écran.

— Sur un moniteur noir et blanc, les couleurs apparaissent comme six niveaux de gris.

— Comme INK, PAPER est à la fois une fonction et une instruction qui renvoie la couleur sélectionnée pour le fond avec PRINT PAPER.

Exemple :

Ce programme change la couleur de l'écran, ligne par ligne :

```
50 INK 7
100 PAPER 0
110 A=0
120 CLS
130 FOR I=0 TO 230 STEP 10
140 PAPER A
150 PRINT@ 3,I
160 A=A+1
170 IF A=8 THEN A=0
180 NEXT I
190 PAPER 0
```

On peut utiliser la fonction PAPER pour imprimer les numéros des couleurs, en ajoutant :

PROTECT

Syntaxe : PROTECT <couleur>

Application : Désactive la banque de couleur spécifiée. <couleur> peut être un numéro ou un nom, comme pour INK et PAPER. Une fois désactivée, la couleur ne peut plus être utilisée pour écrire. Toute portion de l'écran contenant cette couleur ne peut plus être effacée, même par CLS. PROTECT Ø réinitialise le mode standard du Lynx.

REMARQUES :

— Si une couleur de l'écran provient du mélange de deux autres, CLS efface celle des deux dont la banque n'est pas désactivée par PROTECT.

— On peut protéger une couleur composée, dans ce cas les deux banques concernées sont désactivées.

— Si l'on tente d'écrire dans une couleur composée, dont l'une des primaires est désactivée, seule apparaît à l'écran la couleur non protégée.

— PROTECT 7 désactive simultanément les trois banques.

— L'emploi de PROTECT accélère la vitesse d'affichage. Si deux banques de couleurs sont désactivées, celui-ci est à peu près deux fois plus rapide (ce qui n'est pas un luxe !).

— Comme lors de l'utilisation du code VDU 21 de surimpression, on peut grâce à PROTECT écrire à l'écran sans perturber l'affichage existant (graphique ou texte).

Exemple 1 :

Ce programme remplit l'écran, puis protège son contenu de toute tentative d'effacement :

```
100 REM PROTECT
110 PAPER 0
120 CLS
130 INK 2
140 FOR I=1 TO 960
150 PRINT"A";
160 NEXT I
170 PROTECT 2
180 INK 7
190 VDU 24
200 PRINT@ 3,60;"TEXTE EN SURIMPRESSION";
210 VDU 25
220 PRINT@ 3,235;"FIN";
```

Une couleur d'affichage est spécifiée (2-Rouge), l'écran est rempli de " A" et la couleur 2 désactivée. On demande ensuite une autre couleur (7-Blanc), et un petit message imprimé en surimpression pour illustrer le procédé.

REMARQUE: PROTECT est particulièrement adapté au mélange texte/graphique, voyez paragraphe 4.1.4.

Exemple 2 :

Cet autre exemple montre comment on peut employer PROTECT pour *effacer sélectivement* le contenu de l'écran : Tous les caractères non protégés sont supprimés, ceux dont la banque de couleur est désactivée restant affichés.

```
100 REM EFFACEMENT SELECTIF
110 WINDOW 3,123,5,245
115 PAPER 0
120 CLS
130 FOR I=1 TO 114
140 FOR J=1 TO 7
150 INK J
160 PRINT"A";
170 NEXT J
180 NEXT I
190 INK 7
200 WINDOW 3,123,215,245
210 VDU 23
220 INPUT"LAQUELLE DES SIX COULEURS DESIREZ-VOUS....PROTEGER
(1-6) ";C
230 IF C>7 OR C<1 THEN GOTO 210
280 PROTECT C
290 CLS
300 PROTECT 0
320 PRINT@ 3,235;
```

Explication :

120,180 : Remplissage de l'écran, dans toutes les couleurs disponibles.

200,230 : Choix de la couleur à protéger.

280,290 : Protection de la couleur et effacement de l'écran.

TEXT

Syntaxe : TEXT

Application : Cette instruction effectue en série les commandes suivantes : PROTECT 0, INK 4, PAPER 0, CLS, PROTECT 3. L'écran est effacé, et deux couleurs désactivées. L'affichage en vert sur noir est deux fois plus rapide qu'en mode normal. L'utilisation de TEXT est particulièrement adaptée à la visualisation sur un moniteur monochrome.

REMARQUES :

— PROTECT 0 annule ce mode.

— En TEXT, l'utilisation de deux pages-écran séparées est rendue possible par l'intermédiaire de la banque de couleur "vert alternatif" (voir *exemple 2*).

Exemple 1 : Comparaison des vitesses d'affichage, en mode normal et en mode TEXT.

```
100 REM TEXTE
105 PROTECT 0
110 VDU 2,0,4,1,7
120 N=1
140 FOR I=1 TO 960
150 PRINT"A";
160 NEXT I
165 IF N=2 THEN END
170 PRINT@ 3,225;"POUR COMPARER AVEC LA VITESSE EN MODE TEXT,
TAPEZ UNE TOUCHE";
180 A=GETN
190 TEXT
200 N=2
210 GOTO 140
```

Exemple 2 : Constitution de deux pages-écran.

Nous avons vu dans le paragraphe 3.1.3 qu'il existe une quatrième banque de couleurs, appelée "vert alternatif", et située dans la même page-

mémoire que le vert à l'adresse 4Ø96Ø. On peut, par l'intermédiaire du pointeur de la banque "vert", forcer le stockage des données destinées à cette dernière dans la banque vert alternatif. On disposera de cette manière de deux écrans, en affichant au choix le contenu de l'une ou l'autre banque. Ce choix s'effectue à l'aide du port 128 du Z8Ø.

Voici la méthode à suivre :

TEXT	sélection de l'affichage en vert sur noir.
DPOKE 25234,4Ø96Ø	détourne les données destinées à l'écran vers la banque vert alternatif ; le curseur n'est plus affiché à l'écran.

ÉCRITURE PAGE 2 (Sans la voir)

DPOKE 25234,49152	resélectionne l'affichage standard (c'est-à-dire l'envoi des données vers la première page-écran). Rappelons que 49152 est la valeur normalement retournée par DPEEK (25234).
OUT 128,16	fait apparaître la deuxième page-écran, qui disparaît aussitôt en mode direct : Seul, le mode programme permettra de l'exploiter complètement.
OUT 128,4	retour à la première page-écran (inutile en mode direct).

Un exemple est indispensable pour bien comprendre le principe de fonctionnement :

```
100 REM PAGE 2
110 TEXT
120 DPOKE 25234,40960
130 VDU 4,24
150 PRINT"CECI EST LA DEUXIEME PAGE-ECRAN"
160 VDU 25
170 PRINT@ 3,200;"POUR REVENIR A L'AFFICHAGE NORMAL, TAPEZUNE
TOUCHE"
180 DPOKE 25234,49152
190 VDU 23,24
210 PRINT"PAGE-ECRAN NORMALE"
220 VDU 25
230 PRINT@ 3,200;"POUR AFFICHER LA DEUXIEME PAGE, TAPEZ...UNE
TOUCHE"
240 X=GETN
250 OUT 128,16
260 X=GETN
270 OUT 128,4
280 GOTO 240
```

Explication :

110 : Sélection de l'affichage TEXT (vert sur noir).

120 : Les données à afficher sont dirigées vers la deuxième page-écran.

130 : Effacement de la deuxième page, et passage en mode double taille.

150,170 : Affichage d'un commentaire sur la deuxième page.

180 : Les données sont réorientées vers l'écran normal.

190;230 : Impression d'un commentaire sur la première page.

240,280 : Passage d'une page à l'autre par une saisie clavier.

REMARQUES :

— Tout envoi de données vers l'écran ayant pour effet de repasser à la première page, OUT 128,16 doit être suivi d'une suspension de l'affichage : Saisie clavier, calculs ou pause.

— On peut imprimer indifféremment du graphique ou du texte sur les deux pages, l'écriture sur la page deux se faisant toujours "en aveugle".

3.1.4. L'éditeur

On appelle "Éditeur" l'ensemble des fonctions disponibles au clavier pour la mise au point des programmes, ou des lignes de commandes. Celui du Lynx, assez performant, autorise l'insertion de caractères dans une ligne, la recopie de lignes de programme, et la gestion des déplacements du curseur. Précisons toutefois que nous avons à faire à un éditeur "de ligne" : Chaque ligne est chargée dans une mémoire-tampon appelée "buffer" avant sa prise en compte par la machine.

Les commandes de l'Éditeur

CONTROL **E** Sélection du numéro de ligne à éditer. La ligne demandée est chargée dans le buffer, et affichée à l'écran avec le

curseur sur le premier caractère. Si le numéro entré ne correspond pas une ligne existante, une erreur est provoquée.

CONTROL **Q**	Affiche le contenu du Buffer à l'écran (rappelle la dernière ligne éditée, ou la dernière commande entrée en mode direct).
→	Déplace le curseur d'un espace vers la droite.
←	Déplace le curseur d'un espace vers la gauche.
↑	Place le curseur au début de la ligne.
↓	Place le curseur à la fin de la ligne.
DELETE	Efface le caractère placé avant le curseur et déplace celui-ci d'un espace vers la gauche.
RETURN	Valide la ligne modifiée (le Buffer est recopié en mémoire).

REMARQUES :

— Le Lynx est toujours en mode insertion, et la frappe d'un caractère repousse d'un cran toute la ligne à droite du curseur.

— Seule, l'instruction DEL <N° de ligne> permet la suppression de lignes de programme.

Pour recopier une ligne de programme, il faut :

- Taper CONTROL E, suivi du numéro de la ligne à recopier.
- Remplacer le N° de la ligne par le nouveau.
- Valider avec RETURN.

3.1.5. La gestion de l'écran par les ports du Z 80

Les ports 134, 135 et 137 du Z 8Ø sont les voies de passage vers l'écran du système d'entrées-sorties du Lynx. Une modification du statut de ces ports peut être employée pour agir sur le format de l'affichage vidéo.

PORT 137

OUT 137,13 fait disparaître fugitivement l'écran. On peut s'en servir pour simuler un clignotement de l'affichage :

```
100 DEFPROC CLIGNOTEMENT (N,T)
105 A=0
110 WHILE A<N
120 OUT 137,13
130 PAUSE T
140 VDU 14
150 PAUSE T
160 VDU 15
165 A=A+1
170 WEND
180 ENDPROC
```

REMARQUE : Le programme s'exécute par PROC CLIGNOTEMENT (N, T). N doit contenir le nombre de clignotements désiré, et T le temps les séparant.

Explication :

11Ø : Boucle de N clignotements.

12Ø : Disparition de l'écran.

13Ø : Fixe en mode "disparition" pour un temps T.

14Ø : Une opération d'affichage fait réapparaître le texte.

15Ø : Réinitialise le curseur en mode normal.

PORT 134

Ce port gère la position physique du texte par rapport aux limites de l'écran. Deux cas sont possibles :

— *Initialisation avec OUT 134,13* :
Dans ce cas, OUT 135,X avec X compris entre Ø et 32, déplace tout le contenu de l'écran de gauche à droite jusqu'au retour à la position d'origine. Si X va de Ø à 224 par pas de 32, l'écran est déplacé de haut en bas sur sept lignes, ligne par ligne.

— *Initialisation avec OUT 134,7* :
Cette fois, OUT 135,X avec X compris entre 1 et 68, déplace le contenu de l'écran de haut en bas par pas de trois pixels, jusqu'au retour à la position d'origine.

REMARQUE: On pourra se servir utilement de ces fonctions dans des jeux pour faire vibrer l'écran ou défiler du texte et des dessins (scrolling).

***Exemple*:**

```
100 REM  PORTS Z80
119 CLS
120 PRINT"L'INSTRUCTION OUT AUTORISE L'UTILISATIONDIRECTE DES
PORTS DU Z80 POUR :"
130 PRINT
140 PRINT"FAIRE DISPARAITRE L'ECRAN !"
145 PAUSE 2000
150 T=2
160 PROC DISPARITION (T)
170 PRINT
180 PRINT"OU ENCORE CLIGNOTER..."
190 T=25,N=20
210 PROC CLIGNOTEMENT (N,T)
220 PRINT
230 PRINT"L'AFFICHAGE PEUT DEFILER EN TOUS SENS :"
235 PAUSE 3000
240 T=10
250 PROC HORIZONTAL (T)
255 PAUSE 3000
260 PROC VERTICAL (T)
270 PRINT
275 PRINT"POUR LES JEUX, UNE EXPLOSION PEUT FAIRE TREMBLER
L'ECRAN"
276 PAUSE 3000
280 T=10
290 PROC TREMBLOTEMENT (T)
300 PRINT
310 PRINT"OU MEME PROVOQUER UN SEISME !"
315 PAUSE 2000
320 T=5
330 PROC TREMBLEMENT TERRE (T)
340 PRINT
350 PRINT"TERMINE"
360 END
1000 DEFPROC DISPARITION (T)
1010 FOR I=1 TO T*1000
1020 OUT 137,13
1030 NEXT I
1040 ENDPROC
1060 DEFPROC CLIGNOTEMENT (N,T)
1070 FOR I=1 TO N
1080 OUT 137,13
1090 PAUSE 100*T
1110 OUT 137,0
1115 PAUSE 100*T
```

```
1120 NEXT I
1130 ENDPROC
1150 DEFPROC HORIZONTAL (T)
1160 OUT 134,13
1170 FOR I=1 TO 32
1180 OUT 135,I
1190 PAUSE 100*T
1210 NEXT I
1220 ENDPROC
1240 DEFPROC VERTICAL (T)
1250 OUT 134,7
1260 FOR I=15 TO 68
1270 OUT 135,I
1280 PAUSE 100*T
1300 NEXT I
1310 ENDPROC
1330 DEFPROC TREMBLOTEMENT (T)
1340 OUT 134,13
1350 FOR I=1 TO T*10
1360 OUT 135,RAND(4)
1370 NEXT I
1380 OUT 135,0
1390 ENDPROC
1410 DEFPROC TREMBLEMENT TERRE (T)
1420 OUT 134,5
1440 OUT 135,44
1450 PAUSE 200
1460 OUT 135,0
1470 OUT 134,13
1480 FOR I=1 TO T*10
1490 OUT 135,RAND(8)
1500 PAUSE 250
1510 OUT 135,RAND(7)*32
1520 NEXT I
1530 OUT 135,0
1540 ENDPROC
```

Explication : Le programme utilise successivement les ports cités précédemment, en les incluant dans des boucles paramétrées. On peut réutiliser n'importe laquelle de ces procédures dans un programme, en assignant à T et à N les valeurs désirées.

3.2. Les jeux de caractères

3.2.1. Généralités

Le Lynx possède deux jeux de caractères, composés comme suit :

— Un jeu standard, correspondant à la norme Américaine ASCII et comprenant les caractères alphanumériques et les ponctuations. Leurs codes vont de 32 ("Espace") à 127 (2e moitié de "copyright"), soit 96 caractères.

— Un deuxième jeu, composé de 26 mosaïques graphiques dont les codes sont compris entre 224 ("espace") et 249 (dernière partie du signe Lynx).

— Enfin, une copie du jeu alphanumérique standard est accessible par les codes 128 à 223.

Tous ces caractères peuvent s'obtenir par le clavier, CONTROL 1 agit comme une bascule et assigne aux touches une mosaïque graphique à la place du caractère normal. Une deuxième pression sur CRTL 1 réinitialise le clavier.

3.2.2. La génération de caractères

Les mosaïques graphiques et le jeu ASCII sont assez mal adaptés à l'écriture en Français, qui nécessite l'usage de minuscules accentuées ; il en va de même pour les jeux sophistiqués, utilisant des caractères spéciaux tels que notes de musiques, cartes à jouer ou invaders.

Ce chapitre a pour objet de pallier à ces carences, d'une part en expliquant le principe de création d'un caractère, et d'autre part à l'aide d'un programme très sophistiqué de reconfiguration des jeux.

La manipulation de caractères implique l'accès direct en mémoire, selon quelques règles assez simples :

Principes de fonctionnement

Sur le Lynx, un pointeur informe le Basic de l'adresse en ROM où démarre la table des caractères, dont le code est supérieur à 128. En modifiant le

contenu de ce pointeur (GRAPHIC, voir § 2.4.3), on peut utiliser n'importe quelle zone de la mémoire vive pour y écrire ses propres caractères, auxquels on accèdera par les mêmes codes. Le seul inconvénient est la perte des mosaïques graphiques habituellement accessibles au clavier.

Un caractère est inclus dans une matrice de 6 × 1Ø, chaque ligne horizontale de cette matrice correspondant à un codage en binaire sur un octet, dont seuls les six bits les moins significatifs sont utilisés. Chaque caractère est donc codé sur dix octets qui se suivent en mémoire. Un programme générateur réalise toutes les modifications d'un caractère en lisant et en écrivant dans les six adresses appropriées.

Rappels sur la conversion Binaire-Décimal

Soit le nombre binaire B, avec B=XXXXXXXX, X ne pouvant prendre que la valeur 1 ou Ø. Si l'on considère que le bit de gauche est le plus significatif, la valeur décimale D est égale à :

$$D=X*2^7+X*2^6+X*2^5+X*2^4+X*2^3+X*2^2+X*2^1+X*2^0$$

BIN permet un conversion directe, avec D=BIN(B).

Exemple :

Matrice						*Adresse*	*Contenu décimal*	*Contenu binaire*
						798	Ø	ØØ ØØØØØØ
			*			799	4	ØØ ØØØ1ØØ
		*		*		8ØØ	1Ø	ØØ ØØ1Ø1Ø
	*				*	8Ø1	17	ØØ Ø1ØØØ1
	*				*	8Ø2	17	ØØ Ø1ØØØ1
	*	*	*	*	*	8Ø3	31	ØØ Ø11111
	*				*	8Ø4	17	ØØ Ø1ØØØ1
	*				*	8Ø5	17	ØØ Ø1ØØØ1
						8Ø6	Ø	ØØ ØØØØØØ
						8Ø7	Ø	ØØ ØØØØØØ

Explications :

— Le caractère "A" est codé à partir de l'adresse 798 de la ROM. La fonction LETTER permet de connaître cette adresse en fonction du code du caractère : LETTER (<Code>).

— L'adresse 798 correspond à une ligne vide de la matrice, et contient l'octet de valeur Ø.

— L'adresse 8Ø3 correspond à cinq points consécutifs de la 6e ligne de la matrice, et contient l'octet de valeur 31 soit ØØØ11111 en binaire.

— Rappelons que le codage s'effectue de droite à gauche, en commençant par le bit de poids le plus faible.

— Les deux bits de gauche sont inutilisés sur le Lynx.

Création d'un caractère

Il est possible de créer ses caractères sans passer par un programme complexe, en réservant au préalable une zone-mémoire juste suffisante pour les quelques caractères à définir. L'intérêt de ce procédé est que l'on peut redéfinir quelques caractères répondant à un besoin particulier dans un programme, sans être obligé de charger un jeu complet à chaque utilisation.

Voici la marche à suivre :

— Définir sur un papier les caractères, en dessinant leur matrice sous forme d'un damier de 6 × 1Ø, et en noircissant les cases correspondant au dessin de chacun d'entre eux.

— Pour chaque caractère, on déduit le nombre binaire correspondant au contenu de chaque ligne horizontale de la matrice (Ø pour une case blanche et 1 pour une case noire).

— On calcule le nombre d'adresses (N) qu'il est nécessaire de réserver pour y placer les caractères :

```
N=<nombre de caractères>*1Ø
```

— Il faut déplacer le HIMEM du nombre d'adresses calculé, afin de placer les caractères dans une zone protégée :

```
RESERVE HIMEM-N
```

— GRAPHIC doit ensuite pointer sur la zone réservée :

DPOKE GRAPHIC,HIMEM

Une fois cette opération effectuée, les codes de 128 à 249 correspondent aux adresses suivant HIMEM. Le jeu graphique n'est donc plus accessible, et le curseur clignotant (code 249) est remplacé par un caractère aléatoire. On peut restaurer un curseur avec :

CCHAR 32*256+42.

— Enfin, on doit entrer à partir de HIMEM, à l'aide de POKE, la conversion en décimal (BIN) du nombre binaire associé à chaque ligne horizontale de la matrice, adresse par adresse et caractère par caractère.

Exemples :

```
100 REM NOTE
110 CLS
120 RESERVE HIMEM-10
130 DPOKE GRAPHIC,HIMEM
140 CCHAR 256*32+35
150 FOR I=0 TO 9
160 READ R
170 POKE LETTER(128)+I,BIN(R)
180 NEXT I
185 GOTO 290
190 DATA 000000
200 DATA 000011
210 DATA 000010
220 DATA 000011
230 DATA 000010
240 DATA 011010
250 DATA 111110
260 DATA 111100
270 DATA 011000
280 DATA 000000
290 REM DEMO
300 FOR I=1 TO 400
310 PRINT CHR$(128);"  ";
320 NEXT I
330 END
```

Explications :

12∅ : On décale HIMEM de dix adresses pour y placer le caractère.

13∅ : GRAPHIC pointe sur la zone-mémoire démarrant à HIMEM.

140 : Le curseur est reconstitué.

190,250 : Les DATA successifs contiennent le dessin du caractère en binaire sur six bits (les deux bits les plus significatifs peuvent être omis).

150,180 : Boucle lisant un à un les DATA avec READ. Les lignes de DATA sont converties en décimal par BIN, puis placées aux dix adresses correspondant au caractère de code 128. Rappelons que LETTER(C) renvoie la première adresse du caractère de code C.

290,330 : Affichage du caractère créé.

Matrice	*Adresse*	*Contenu décimal*	*Contenu binaire*
`      `	65518	0	00 000000
`    **`	65519	3	00 0000**11**
`    * `	65520	2	00 0000**1**0
`    **`	65521	3	00 0000**11**
`    * `	65522	2	00 0000**1**0
` ** * `	65523	26	00 0**11**0**1**0
`***** `	65524	62	00 **11111**0
`****  `	65525	60	00 **1111**00
` **   `	65526	24	00 0**11**000
`      `	65527	0	00 000000

REMARQUES :

— RESERVE HIMEM-10 décale le HIMEM de 10 adresses à chaque utilisation du programme, ce qui peut se solder par une "OUT OF MEMORY ERROR" après un certain nombre d'exécutions. Dans un programme opérationnel, il vaut mieux remplacer la ligne 120 par :

```
120 RESERVE 65518          (Lynx 96K.)

120 RESERVE 40940          (Lynx 48K.)
```

— On peut éviter la perte des mosaïques graphiques, en réservant suffisamment de mémoires pour tous les caractères dont le code est compris entre 128 et 249, puis en recopiant le jeu alternatif dans cette zone réservée. C'est la méthode utilisée dans le programme GENECAR :

```
1000 RESERVE HIMEM-1219
1010 DPOKE GRAPHIC,HIMEM
1020 J=HIMEM
1030 FOR I=468 TO 1687
1040 POKE J,PEEK(I)
1050 J=J+1
1060 NEXT I
```

Explication :

Ce programme copie le contenu des adresses 468 à 1687 (partie de la ROM contenant les caractères de code 128-249) dans la zone réservée au-dessus de HIMEM. Le curseur en vidéo inverse est automatiquement réinitialisé. Les caractères sont tous modifiables et remplaçables, LETTER renvoie leurs premières adresses. Sur le Lynx 96K., on peut remplacer HIMEM-1219 (ligne 1ØØØ) par 643ØØ et par 397ØØ sur la version 48K.

3.2.3. Le programme GENECAR

Les procédures de modifications de caractères exposées ci-dessus sont idéales lorsque vous n'avez à en redessiner que quelques uns, pour application particulière. Mais si vous voulez reconfigurer un jeu complet, puis le sauvegarder sur K7 pour réutilisation ultérieure, il est indispensable de passer par un programme plus complet. Voici ce que vous pourrez faire avec GENECAR :

1. Afficher à l'écran le jeu complet, ainsi qu'une matrice agrandie.

2. Choisir un caractère du jeu, en se déplaçant dans celui-ci avec un curseur spécial.

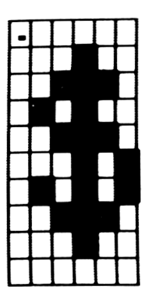
3. Transférer le caractère choisi dans la matrice, le modifier en dessinant avec les quatre flèches, puis replacer ce caractère n'importe où dans le jeu.

```
GENERATEUR DE CARACTERES ©BURP

 ! "#█%&' () *+, -./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\] @ £abcdefghijklmno
pqrstuvwxyz→←↑↓ ...LYNX

CODE DU CARACTERE SOUS LE CURSEUR :132

SA PREMIERE ADRESSE EN MEMOIRE :64340
```



```
→←↑↓     : MOUV. MATRICE
<>       : MOUV. JEU CAR
BARRE    : ON/OFF POINT
RETURN   : MATRICE→JEU
C        : JEU→MATRICE
DEL      : EFFACER MAT.
S        : SAVE JEU/K7
L        : LOAD JEU DU K7
Q        : QUIT
J        : RESET JEU
```

4. Transférer des caractères d'un emplacement à l'autre dans le jeu.

5. Créer entièrement un caractère, puis le placer à n'importe quel endroit du jeu.

6. Sauvegarder et recharger un jeu sur K7.

7. Sortir du programme sans perdre le jeu.

REMARQUES :

— Une fois le jeu sauvegardé, on peut le recharger à partir d'un programme Basic en entrant (Lynx 96K.) :

```
10 RESERVE 64300
20 DPOKE GRAPHIC,HIMEM
30 MLOAD"JEU"
```

Sur le Lynx 48K., il faut écrire :

```
10 RESERVE 39700
```

Le jeu est chargé dans une zone au-dessus de HIMEM, et utilisable par l'intermédiaire des codes compris entre 128 et 249.

— La modification par GENECAR des mosaïques graphiques autorise la création de caractères directement accessibles au clavier par CTRL 1, à la place de l'alphabet standard.

— Sur la version 48K. du Lynx, on doit remplacer les lignes 22Ø, 1Ø6Ø, 268Ø et 298Ø de GENECAR par :

```
 22Ø RESERVE 397ØØ
1Ø6Ø LET c=128,a=397ØØ
268Ø PRINT "TAPEZ: D 9B14 9FCE Ø 'jeu'"
298Ø P=3,p=35,c=128,N=3,n=35,a=397ØØ
```

Commandes disponibles :

< >	Déplacements dans le jeu de caractères.
C	Copie dans la matrice le caractère pointé par le curseur "jeu".
RETURN	Copie à la position du jeu pointée par le curseur le caractère se trouvant dans la matrice.
↓ ↑ ← →	Déplacements dans la matrice.
BARRE	Trace/efface un point de la matrice (bascule).
DELETE	Efface la matrice.
S	Sauvegarde sur K7.
L	Chargement d'un jeu sur K7.
Q	Quitte le programme en conservant le jeu créé en mémoire.
J	Réinitialise le programme avec le jeu standard.

```
100 REM GENECAR BVRP
110 GOSUB  LABEL AIDES
120 GOSUB  LABEL INITIALISATION
130 GOSUB  LABEL VARIABLES
140 GOSUB  LABEL ECRAN
150 GOSUB  LABEL EFFACER CADRE
160 GOSUB  LABEL INSCRIPTIONS
170 GOSUB  LABEL AFFICHAGE JEU
180 GOSUB  LABEL INSCRIPTION 2
190 GOSUB  LABEL BOUCLE
200 END
210 LABEL INITIALISATION
220 RESERVE 64300
230 DPOKE GRAPHIC,HIMEM
240 LET J=HIMEM
250 FOR I=468 TO 1687
260   POKE J,PEEK(I)
270   LET J=J+1
280 NEXT I
290 CCHAR 256*32+239
300 RETURN
310 LABEL ECRAN
320 PROTECT 0
330 PAPER A
340 WINDOW 3,123,5,245
350 CLS
360 PAPER B
370 PRINT  @ 3,5
380 PRINT  @ 3,25
390 PRINT  @ 3,35
400 PRINT  @ 3,45
410 PRINT  @ 3,65
420 PRINT  @ 3,85
430 PAPER A
440 INK D
450 FOR J=125 TO 225 STEP 10
460   FOR I=5 TO 41
470     DOT I,J
480   NEXT I
490 NEXT J
500 FOR J=5 TO 41 STEP 6
510   FOR I=125 TO 225
520     DOT J,I
530   NEXT I
540 NEXT J
550 PROTECT B
560 RETURN
570 LABEL EFFACER CADRE
580 PROTECT 0
590 PAPER B
600 WINDOW 50,123,125,225
610 VDU 23
620 FOR I=1 TO 10
```

```
630   PRINT
640 NEXT I
650 PROTECT B
660 RETURN
670 LABEL INSCRIPTIONS
680 INK D
690 PRINT  @ 15,15;"GENERATEUR DE CARACTERES   BVRP";
700 VDU 23
710 PRINT CHR$(123);CHR$(124);CHR$(125);CHR$(126);"     : MOUV. MATRICE  ";
720 PRINT "<>        : MOUV. JEU CAR"
730 PRINT "BARRE    : ON/OFF POINT"
740 PRINT "RETURN   : MATRICE";CHR$(123);"JEU"
750 PRINT "C        : JEU";CHR$(123);"MATRICE"
760 PRINT "DEL      : EFFACER MAT."
770 PRINT "S        : SAVE JEU/K7"
780 PRINT "L        : LOAD JEU DU K7"
790 PRINT "Q        : QUIT"
800 PRINT "J        : RESET JEU";
810 PROTECT D
820 INK 7
830 PRINT  @ P,p;CHR$(239);
840 GOSUB  LABEL BRUIT
850 RETURN
860 LABEL BOUCLE
870 PRINT  @ T,t;T$
880 LET X=GETN
890 BEEP 100,10,63
900 IF X=44 OR X=46 THEN  GOSUB  LABEL JEU
910 IF X=74 OR X=106 THEN  GOSUB  LABEL RESET
920 IF X=81 OR X=113 THEN  GOSUB  LABEL QUIT
930 IF X=67 OR X=99 THEN  GOSUB  LABEL COPY
940 IF X=8 THEN  GOSUB  LABEL DELETE
950 IF X=13 THEN  GOSUB  LABEL RETOUR JEU
960 IF X=10 OR X=11 OR X=12 OR X=22 THEN  GOSUB  LABEL CREATION
970 IF X=32 THEN  GOSUB  LABEL ON/OFF
980 IF X=83 OR X=115 THEN  GOSUB  LABEL SAVE
990 IF X=76 OR X=108 THEN  GOSUB  LABEL LOAD
1000 GOTO 880
1010 LABEL VARIABLES
1020 LET m=3,M=120,h=35,H=55
1030 LET P=3,p=35
1040 LET A=0,B=1,C=2,D=3
1050 LET N=P,n=p
1060 LET c=128,a=64300
1070 DIM C(10)
1080 LET T=3,t=125,F=3,f=125
1090 LET T$=CHR$(46)
1100 DIM T(6*10)
1110 FOR I=1 TO 10
1120   FOR J=1 TO 6
1130     LET T(J*10-I)=0
1140   NEXT J
1150 NEXT I
```

```
1160 RETURN
1170 LABEL AFFICHAGE JEU
1180 PROTECT 0
1190 PROTECT B
1200 INK D
1210 LET S=128,s=35
1220 REPEAT
1230   PRINT  @ 3,s;
1240   FOR I=S TO S+39
1250     PRINT CHR$(I);
1260   NEXT I
1270   LET S=S+40,s=s+10
1280 UNTIL s=65
1290 PRINT  @ 3,75;"CODE DU CARACTERE SOUS LE CURSEUR :";
1300 PRINT  @ 3,95;"SA PREMIERE ADRESSE EN MEMOIRE :";
1310 PROTECT D
1320 INK 7
1330 PRINT  @ P,p;CHR$(239);
1340 RETURN
1350 LABEL INSCRIPTION 2
1360 PRINT  @ 108,75;c
1370 PRINT  @ 99,95;a
1380 RETURN
1390 LABEL JEU
1400 IF NOTX=46 THEN  GOTO 1470
1410 IF P=M THEN  GOTO 1440
1420 LET P=P+3
1430 GOTO 1520
1440 IF p=H THEN  LET p=h,P=m
1450 ELSE  LET p=p+10,P=m
1460 GOTO 1520
1470 IF P=m THEN  GOTO 1500
1480 LET P=P-3
1490 GOTO 1520
1500 IF p=h THEN  LET p=H,P=M
1510 ELSE  LET p=p-10,P=M
1520 PRINT  @ N,n;CHR$(32);
1530 PRINT  @ P,p;CHR$(239);
1540 LET N=P,n=p
1550 LET K=(p-35)/10,c=127+(P/3)+(K*40),a=LETTER(c)
1560 GOSUB  LABEL INSCRIPTION 2
1570 RETURN
1580 LABEL CREATION
1590 IF X=11 AND NOTt=125 THEN  LET t=t-10
1600 IF X=10 AND NOTt=215 THEN  LET t=t+10
1610 IF X=12 AND NOTT=18 THEN  LET T=T+3
1620 IF X=22 AND NOTT=3 THEN  LET T=T-3
1630 LET I=(f-115)/10,J=F/3
1640 IF T(J*10-I)=0 THEN  PRINT  @ F,f;" ";
1650 ELSE  PRINT  @ F,f;CHR$(239);
1660 LET I=(t-115)/10,J=T/3
1670 IF T(J*10-I)=0 THEN  PRINT  @ T,t;T$
1680 ELSE  PRINT  @ T,t;CHR$(242);
```

```
1690 LET F=T,f=t
1700 RETURN
1710 LABEL ON/OFF
1720 LET I=(t-115)/10,J=T/3
1730 IF T(J*10-I)=0 THEN  GOTO 1770
1740 PRINT  @ T,t;T$
1750 LET T(J*10-I)=0
1760 RETURN
1770 PRINT  @ T,t;CHR$(242)
1780 LET T(J*10-I)=1
1790 RETURN
1800 LABEL DELETE
1810 FOR I=1 TO 10
1820   PRINT  @ 3,(I*10)+115;"        "
1830   FOR J=1 TO 6
1840     LET T(J*10-I)=0
1850   NEXT J
1860 NEXT I
1870 LET T=3,t=125,F=3,f=125
1880 PRINT  @ T,t;T$
1890 BEEP 100,10,63
1900 RETURN
1910 LABEL COPY
1920 GOSUB  LABEL DELETE
1930 LET K=(p-35)/10
1940 LET c=127+(P/3)+(K*40)
1950 LET a=LETTER(c)
1960 FOR I=1 TO 10
1970   LET C(I)=PEEK(a+I-1)
1980   LET R=32
1990   FOR J=1 TO 6
2000     LET T(J*10-I)=C(I) DIV R
2010     IF T(J*10-I)=1 THEN  PRINT  @ J*3,(I*10)+115;CHR$(239)
2020     IF T(J*10-I)=1 THEN  BEEP 100,10,63
2030     LET C(I)=C(I) MOD R,R=R/2
2040   NEXT J
2050 NEXT I
2060 IF T(1*10-1)=1 THEN  PRINT  @ T,t;CHR$(242);
2070 RETURN
2080 LABEL RETOUR JEU
2090 BEEP 100,10,63
2100 FOR I=1 TO 10
2110   LET R=0
2120   FOR J=1 TO 6
2130     LET R=R+(T(J*10-I)*2**(6-J))
2140   NEXT J
2150   POKE (a+I-1),R
2160 NEXT I
2170 GOSUB  LABEL AFFICHAGE JEU
2180 BEEP 100,10,63
2190 RETURN
2200 LABEL AIDES
2210 WINDOW 3,123,5,245
```

```
2220 PROTECT 0
2230 PAPER 1
2240 CLS
2250 INK 7
2260 VDU 24
2270 PRINT  @ 39,5;"GENECAR BVRP";
2280 PRINT
2290 VDU 25
2300 GOSUB  LABEL BRUIT
2310 PRINT  @ 3,55;
2320 PRINT "GENECAR EST UN UTILITAIRE PERMETTANT LA CREATION  D'UN JEU  PERSONNA
LISE, QU'ON PEUT UTILISER EN PLUS DU JEU NORMAL. ON Y ACCEDE GRACE AUX CODES COM
PRIS  ENTRE 128 ET 247."
2330 PRINT
2340 PRINT "ON PEUT MODIFIER, RECREER OU SIMPLEMENT DEPLACER DES CARACTERES DU J
EU AFFICHE. POUR CE FAIRE, UNE MATRICE DE 6X10 SERT DE TRAME POUR LA VISUALISATI
ON DE  TOUS LES POINTS DU CARACTERE."
2350 PRINT
2360 PRINT "UNE  FOIS LE JEU  CREE, UNE  PROCEDURE  PERMET DE  QUITTER  LE  PROG
RAMME POUR  L'UTILISER DIRECTEMENT OU SA SAUVEGARDE SUR K7."
2370 PRINT
2380 PRINT "POUR COMMENCER, TAPEZ UNE TOUCHE..."
2390 LET Z=GETN
2400 GOSUB  LABEL BRUIT
2410 CLS
2420 VDU 24
2430 PRINT  @ 3,50;"PATIENCE....."
2440 VDU 25
2450 RETURN
2460 LABEL QUIT
2470 GOSUB  LABEL BRUIT
2480 WINDOW 3,123,5,245
2490 PROTECT 0
2500 PAPER 0
2510 CLS
2520 INK 7
2530 NEW
2540 END
2550 LABEL LOAD
2560 GOSUB  LABEL EFFACER CADRE
2570 PRINT "POSITIONNER LA K7 "
2580 PRINT
2590 PRINT "METTEZ SUR PLAY ..."
2600 PRINT
2610 MLOAD "JEU"
2620 GOSUB  LABEL EFFACER CADRE
2630 GOSUB  LABEL INSCRIPTIONS
2640 GOSUB  LABEL AFFICHAGE JEU
2650 RETURN
2660 LABEL SAVE
2670 GOSUB  LABEL EFFACER CADRE
2680 PRINT "TAPEZ:D FB2C FFE6 0 'JEU'";
2690 PRINT
```

```
2700 PRINT "METTEZ LE K7 EN RECORD"
2710 PRINT "PUIS VALIDER PAR <RETURN>";
2720 PRINT
2730 PRINT "POUR LE PROGRAMME TAPEZ :";
2740 PRINT
2750 PRINT "J PUIS <RETURN>"
2760 PRINT "PUIS GOTO 2810"
2770 INK D
2780 WINDOW 50,123,215,225
2790 VDU 23
2800 MON
2810 WINDOW 50,123,125,225
2820 INK 7
2830 GOSUB  LABEL EFFACER CADRE
2840 GOSUB  LABEL INSCRIPTIONS
2850 RETURN
2860 LABEL BRUIT
2870 FOR J=0 TO 10
2880   BEEP RAND(50)+50,40,63
2890 NEXT J
2900 RETURN
2910 LABEL RESET
2920 PRINT  @ P,p;CHR$(32);
2930 PRINT  @ 108,75;"    "
2940 PRINT  @ 99,95;"      "
2950 GOSUB  LABEL BRUIT
2960 GOSUB 240
2970 GOSUB  LABEL BRUIT
2980 LET P=3,p=35,c=128,a=64300,N=3,n=35
2990 GOSUB  LABEL DELETE
3000 GOSUB  LABEL AFFICHAGE JEU
3010 GOSUB  LABEL INSCRIPTION 2
3020 GOSUB  LABEL BRUIT
3030 RETURN
```

Structure et explications du programme :

GENECAR est constitué d'une succession de routines, appelées en série par le programme-source (1ØØ-2ØØ) :

Nom du label	*Fonction*
AIDES	Affichage des commentaires et explications.
INITIALISATIONS	Copie des caractères alphanumériques et graphiques dans une zone réservée de la mémoire.

VARIABLES	Initialisation des diverses variables du programme.
ÉCRAN	Présentation-écran de base.
EFFACER	Création d'une fenêtre-texte pour l'affichage des commandes disponibles, et effacement de cette fenêtre.
INSCRIPTIONS	Affichage des commandes disponibles.
AFFICHAGE JEU	Affichage du jeu de caractères complet, et initialisation du curseur.
INSCRIPTION 2	Affiche en permanence le code et la première adresse du caractère pointé par le curseur "jeu".
BOUCLE	Boucle de saisie des commandes. Celle-ci appelle elle-même un certain nombre de sous routines, en fonction des commandes entrées.
Sous-routines:	
JEU	Déplacement du curseur à l'intérieur du jeu.
RESET	Réaffiche le jeu standard non modifié.
QUIT	Sortie du programme.
COPY	Copie dans la matrice le caractère pointé dans le jeu.
DELETE	Effacement de la matrice.
RETOUR JEU	Copie dans le jeu du contenu de la matrice.
CREATION	Déplacement à l'intérieur de la matrice.
ON/OFF	Validation ou annulation d'un point de la matrice.
SAVE	Sauvegarde sur cassette le jeu de caractères.
LOAD	Routine de chargement d'un jeu.

Enfin, un LABEL "BRUIT" sonorise les différentes parties du programme.

REMARQUE: Tous les sous-programmes de GENECAR sont cités, mais chacune des routines présentées en appelle elle-même d'autres suivant les besoins du programme.

4

Le graphique et le son

4.1. L'œil du Lynx : le graphique haute-résolution

Le Lynx dispose d'un mode graphique haute-résolution de 256 par 248 points, adressables individuellement. Tout l'écran est accessible, et les huit teintes disponibles sont toujours utilisables pour définir séparément la couleur de chaque point. Un certain nombre d'instruction sont spécifiques au mode graphique, dont la puissante instruction PLOT qui combine le dessin en coordonnées relatives et en coordonnées absolues.

Comme dans la première partie, ces instructions sont détaillées et parfois associées à des sous-programmes d'exemple ou destinés à pallier à l'absence de certaines commandes (cercle, arc, rectangle).

Deux autres chapitres sont consacrés aux possibilités offertes par le mélange texte/graphique, ainsi qu'à l'utilisation du langage-machine.

4.1.1. Les instructions

Toutes les commandes du graphique font référence à la position d'un curseur à l'écran. Ce curseur est différent de celui qu'emploie le mode texte : Il n'apparaît jamais physiquement, sa position est gérée directement en mémoire.

— La coordonnées horizontale (ou abscisse) du curseur doit être comprise entre Ø et 255. Dans la syntaxe des commandes, cette valeur est représentée par X.

— Sa coordonnée verticale, ou ordonnée, entre Ø et 247. Il s'agit de Y dans la syntaxe.

— Le point de coordonnées Ø,Ø ou point d'origine, correspond au coin supérieur gauche de l'écran.

REMARQUES :

— Toutes les fonctions du graphique sont notablement accélérées en mode monochrome (TEXT).

— On appelle "couleur en vigueur" les couleurs d'affichage sélectionnées par INK et PAPER, selon les mêmes règles que celles définies au paragraphe 3.1.3 pour l'affichage texte.

DOT

Syntaxe : DOT X,Y

Application : Trace un point élémentaire, ou pixel, à l'écran à la position X,Y et dans la couleur en vigueur. La nouvelle position du curseur sera X,Y.

REMARQUE : L'emploi de coordonnées hors-limites est autorisé. Si la valeur de X ou Y dépasse 255 ou 247, le point est affiché à l'écran aux coordonnées :

X MOD 255
Y MOD 247

Cette règle fonctionne également si X et Y sont négatifs.

Exemples :

DOT 1ØØ,1ØØ

DOT 5ØØ,1ØØ est équivalent à DOT 245,1ØØ

DOT 1ØØ,–2Ø est équivalent à DOT 1ØØ,227.

Le programme suivant trace un cercle point par point, après définition de son rayon et des coordonnées du centre par l'utilisateur (voir également § 4.1.2).

```
100 REM CERCLE
110 VDU 2,0,1,7
120 WINDOW 3,123,205,245
130 VDU 4,23
140 INPUT"RAYON ";R
150 INPUT"COORDONNEES DU CENTRE ";X,Y
160 FOR I=0 TO 2*PI STEP PI/(R*3.2)
170 x=SIN(I)*R+X
180 y=COS(I)*R+Y
190 DOT x,y
200 NEXT I
210 END
```

DRAW

Syntaxe : DRAW X,Y

Application : Trace une droite dans la couleur en vigueur, à partir de la position du curseur jusqu'au point de coordonnées X,Y. Comme pour DOT, le curseur est déplacé à la position X,Y et l'emploi de valeurs hors limites est autorisé.

Exemple :

```
100 REM DEMO DRAW
110 CLS
120 FOR I=0 TO 2*PI STEP 0.1
130 x=100+(COS(I)*50),y=100+(SIN(I)*50)*1.2
140 DRAW x,y
150 MOVE 100,100
160 NEXT I
```

Explication :

Draw trace successivement à l'intérieur de la boucle, des lignes à partir du centre (1ØØ,1ØØ) vers la périphérie du cercle (x,y).

MOVE

Syntaxe : MOVE X,Y

Application : MOVE déplace le curseur aux coordonnées X et Y, sans tracer aucun point.

Exemple :

```
100 REM POINTILLES
110 CLS
120 FOR Y=0 TO 200 STEP 20
130 Z=0
140 FOR X=0 TO 255 STEP 10
150 IF Z=1 THEN DRAW X,Y
160 ELSE MOVE X,Y
170 Z=NOT Z
180 NEXT X
190 NEXT Y
```

PLOT

Syntaxe : PLOT <mode>,X,Y

Application : Cette instruction puissante autorise cinq types de tracés différents en fonction de <mode> :

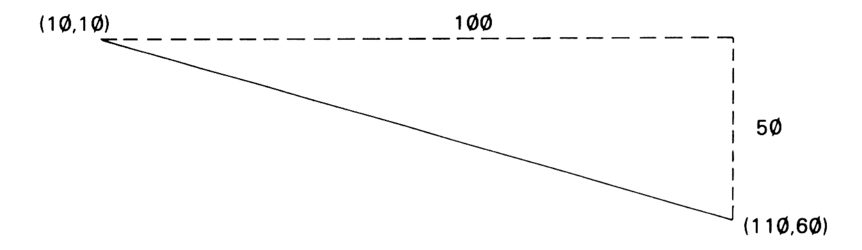
— PLOT Ø est identique à MOVE.

— PLOT 1 est un MOVE relatif. Le curseur est déplacé du *nombre de points* fixé par X et Y, et non au point de coordonnées X,Y. Par exemple, PLOT 1,1ØØ,4 déplace le curseur de 1ØØ pixels horizontalement et de 4 verticalement.

— PLOT 2 est identique à DRAW.

— PLOT 3 est un DRAW relatif. Comme pour PLOT 1, le curseur est déplacé du nombre de points fixé par X et Y, mais une droite est tracée en même temps. Celle-ci représente la diagonale d'un rectangle, dont les coins opposés correspondent respectivement à la position d'origine du curseur et à son point d'arrivée (la longueur de ce rectangle fictif est X, et sa hauteur Y).

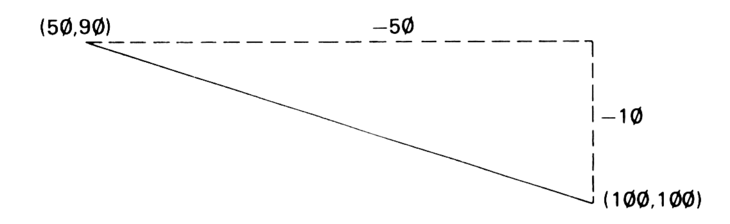
Par exemple, si la position d'origine est 1Ø,1Ø PLOT 3,1ØØ,5Ø peut se schématiser ainsi :



— PLOT 4 est identique à DOT.

REMARQUES :

— Comme pour les autres instructions, on peut spécifier pour PLOT des coordonnées négatives. Dans le cas de PLOT 1 et de PLOT 3, ces coordonnées entraînent un déplacement inversé sur l'échelle des abscisses et des ordonnées. Soit par exemple 1ØØ,1ØØ la position du curseur, PLOT 3,—5Ø,—1Ø aura pour effet de tracer une droite vers le point de coordonnées 5Ø,9Ø.



— PLOT est utile pour paramétrer les instructions, dans la mesure où <mode> peut être une variable.

Exemple :

Le dessin en relatif autorise, en paramétrant les valeurs de X et Y, une modification simple de la taille d'un graphique. Le programme présenté démontre cette possibilité en effectuant par une simple boucle un "zooming" sur le dessin de la lettre "E".

```
100 REM PLOT RELATIF
110 VDU 2,0,1,7,4
140 WINDOW 3,123,205,245
150 GOTO 310
160 REM DESSIN LETTRE
170 MOVE X,Y
180 PLOT 3,80*F,0
190 PLOT 3,0,-20*F
```

```
200 PLOT 3,-60*F,0
210 PLOT 3,0,-20*F
220 PLOT 3,30*F,0
230 PLOT 3,0,-20*F
240 PLOT 3,-30*F,0
250 PLOT 3,0,-20*F
260 PLOT 3,60*F,0
270 PLOT 3,0,-20*F
280 PLOT 3,-80*F,0
290 PLOT 3,0,100*F
300 RETURN
310 REM AFFICHAGE
320 FOR F=0.1 TO 1.3 STEP 0.6
330 X=100*F+INT(F)*20
340 Y=100*F+50*F
350 GOSUB 170
360 NEXT F
370 REM COMMENTAIRE
380 PRINT@ 3,210;
390 PRINT"LA LETTRE E EST AFFICHEE A TROIS POINT DIFFERENTS DE
L'ECRAN EN..MODIFIANT X,Y ET DANS TROIS..TAILLES EN MODIFIANT F"
```

Explication:

1ØØ,15Ø: Initialisations.

16Ø,3ØØ: Sous-programme dessinant en relatif une lettre "E", au point de coordonnées X,Y. La variable F paramètre la taille du dessin.

31Ø,36Ø: Boucle faisant varier F pour modifier emplacement et taille de la lettre "E".

4.1.2. Quelques programmes...

Les quelques programmes qui suivent sont des exemples de ce qu'il est réellement possible de faire à l'aide des instructions du graphique Lynx. Tous sont suffisamment simples pour s'adapter aux besoins du lecteur, ou utilisés comme sous-programmes en supprimant les lignes de commentaires et présentations.

Utilisation des 4 flèches du clavier pour dessiner

```
100 REM DESSIN CLAVIER
110 VDU 2,0,1,7,4
130 WINDOW 3,123,5,205
140 X=127,Y=102,x=X,y=Y
```

```
150 MOVE X,Y
160 DOT X,Y
170 PRINT@ 3,205;"LES 4 FLECHES SERVENT AU DEPLACEMENT....";
175 PRINT@ 3,215;"<SPACE> FIN DE SAISIE...................."
180 PRINT@ 3,225;"X=";x;" ";TAB 10;"Y=";y;" ";
190 A=GETN
195 IF A=32 THEN GOTO 300
200 IF A=22 AND x<>0 THEN LET x=x-1
210 IF A=12 AND x<>255 THEN LET x=x+1
220 IF A=11 AND y<>0 THEN LET y=y-1
230 IF A=10 AND y<>205 THEN LET y=y+1
240 PRINT@ 3,225;"X=";x;" ";TAB 10;"Y=";y;" ";
250 DRAW x,y
260 X=x,Y=y
270 GOTO 190
300 END
```

Explications :

110,180 : Initialisation des variables et présentation écran.

190,270 : Boucle principale servant au dessin, comprenant :

240 : Écriture des coordonnées du curseur à l'écran.

190,230 : Saisie de ses mouvements. Les valeurs limites pour x,y empêchent le dépassement des bords de l'écran.

REMARQUES :

— Les principes de base utilisés sont identiques dans tous les programmes d'aide au dessin, quelle que soit leur complexité.

— Une version de ce programme utilisant les ports de Z 80 pour une saisie plus rapide est proposée au paragraphe 4.1.5.

Dessiner un rectangle

```
100 REM RECTANGLE
105 VDU 2,0,1,7,4
110 WINDOW 3,123,205,245
120 VDU 4,23
130 INPUT"COIN SUP. GAUCHE (X,Y) ";C,c
140 INPUT"LONGUEUR RECTANGLE ";L
150 INPUT"LARGEUR  RECTANGLE ";l
160 PLOT 0,C,c
170 PLOT 3,L,0
180 PLOT 3,0,l
190 PLOT 3,-L,0
200 PLOT 3,0,-l
210 END
```

Explications:

50,120: Initialisations.

130,150: Entrée des caractéristiques du rectangle.

160,200: Dessin du rectangle.

Colorier une zone rectangulaire

```
100 REM ZONE RECTANGULAIRE
105 VDU 2,0,1,7
110 WINDOW 3,123,205,245
120 VDU 4,23
130 INPUT"COIN SUP. GAUCHE (X,Y) ";C,c
140 INPUT"LONGUEUR ZONE ";L
150 INPUT"LARGEUR  ZONE ";l
155 INPUT"COULEUR DU TRAIT (1-7) ";A
160 IF A<1 OR A>7 THEN GOTO 155
170 INK A
180 WHILE l>=c
190 PLOT 0,C,c
200 PLOT 3,L,0
210 c=c+1
220 WEND
230 END
```

Explication: La création de la zone se fait par affichage de lignes horizontales jointives dans la couleur choisie.

Dessiner un cercle, une ellipse, un arc

Aucune instruction n'est prévue pour dessiner ces figures. Les trois procédures ci-dessous simulent ces commandes avec possibilité de passage de variables à partir du programme principal:

— PROC CERCLE (R,X,Y) appelle la procédure "CERCLE", avec R=rayon et X,Y les coordonnées du centre.

```
100 DEFPROC CERCLE (R,X,Y)
160 FOR I=0 TO 2*PI STEP PI/(R*3.2)
170 x=SIN(I)*R+X
180 y=COS(I)*R+Y
190 DOT x,y
200 NEXT I
210 ENDPROC
```

— PROC ELLIPSE (R,r,X,Y) appelle la procédure "ELLIPSE", avec R=rayon horizontal r=rayon vertical et X,Y les coordonnées du centre.

```
100 DEFPROC ELLIPSE (R,r,X,Y)
160 FOR I=0 TO 2*PI STEP PI/((r+R)*2.5)
170 x=SIN(I)*R+X
180 y=COS(I)*r+Y
190 DOT x,y
200 NEXT I
210 ENDPROC
```

— PROC ARC (R,X,Y,O,L) appelle la procédure "ARC", avec R=rayon, X et Y les coordonnées du centre, O le point d'origine à partir duquel démarre le tracé, et L la longueur de l'arc.

* O doit être compris entre Ø et 1. Si l'on assimile le cercle complet au cadran d'une montre, O=Ø correspond à 6 h, Ø.5 est midi, Ø.75 est 9 h et 1 est confondu avec Ø (tour complet).
* L doit aussi être compris entre Ø et 1, L=Ø.5 trace un demi-cercle, Ø.25 un quart de cercle, etc...

```
100 DEFPROC ARC (R,X,Y,O,L)
110 A=O*2*PI
120 B=2*PI*L
160 FOR I=A TO A+B STEP PI/(R*3.2)
170 x=SIN(I)*R+X
180 y=COS(I)*R+Y
190 DOT x,y
200 NEXT I
210 ENDPROC
```

4.1.3. Un programme utile?

Voici un programme qui devrait permettre au lecteur de se familiariser avec les instructions du graphique Lynx. La procédure de dessin d'un arc y est incluse, avec une modification : Le point d'origine O et la longueur L ont été remplacés par le point de départ A et le point d'arrivée B, avec A et B compris entre Ø et 2*PI.

```
100 REM END
110 CLS
120 WINDOW 3,123,205,245
125 REM LETTRE E
130 MOVE 0,40
140 PLOT 3,70,0
150 PLOT 3,0,20
160 PLOT 3,-50,0
170 PLOT 3,0,20
180 PLOT 3,30,0
190 PLOT 3,0,20
200 PLOT 3,-30,0
210 PLOT 3,0,20
220 PLOT 3,50,0
230 PLOT 3,0,20
240 PLOT 3,-70,0
250 PLOT 3,0,-100
255 REM LETTRE N
260 MOVE 85,40
270 PLOT 3,0,100
280 PLOT 3,20,0
290 PLOT 3,0,-70
300 DRAW 137,140
310 PLOT 3,30,0
320 PLOT 3,0,-100
330 PLOT 3,-20,0
340 PLOT 3,0,70
350 DRAW 115,40
360 PLOT 3,-30,0
365 REM EXTERIEUR DE D
370 MOVE 182,40
380 PLOT 3,0,100
390 PLOT 3,30,0
400 PROC ARC (40,212,100,0,PI/2)
410 PLOT 3,0,-20
420 PROC ARC (40,212,80,PI/2,PI)
430 PLOT 3,-30,0
435 REM INTERIEUR DE D
440 MOVE 202,60
450 PLOT 3,0,60
460 PLOT 3,5,0
470 PROC ARC (30,207,90,0,PI)
480 PLOT 3,-5,0
490 END
1100 DEFPROC ARC (R,X,Y,A,B)
1160 FOR I=A TO B STEP PI/(R*3.2)
1170 x=SIN(I)*R+X
1180 y=COS(I)*R+Y
1190 DOT x,y
1200 NEXT I
1210 ENDPROC
```

4.1.4. Le mélange texte-graphique

La plupart des ordinateurs dits "familiaux" du commerce disposent d'un mode graphique séparé du mode texte, avec des zones réservées pour l'un et l'autre. Ce n'est pas le cas du Lynx, dont l'affichage est prévu pour fonctionner dans tous les modes sur un même écran :

PRINT @ autorise le placement du texte sur un graphique, avec une résolution horizontale de deux pixels et verticale de un pixel !

Il peut toutefois s'avérer indispensable de disposer de la possibilité de dissocier texte et graphique, ne serais-ce que pour pouvoir les effacer séparément.

Pour ce faire, nous proposerons trois méthodes assez faciles à mettre en œuvre :

1. Créer deux pages-écran, l'une pour le graphique et l'autre pour du texte. On peut effacer et écrire séparément sur les deux pages en dirigeant le pointeur de la banque "vert" vers le vert alternatif. Une fois remplies, les pages s'affichent de manière instantanée (cf. § 3.1.3).
2. Sélectionner une partie de l'écran pour le graphique, en dimensionnant la zone texte avec WINDOW.
   Par exemple, WINDOW 3,123,2Ø5,245 réserve une fenêtre de quatre lignes de haut pour le texte en bas d'écran, et laisse 2Ø5 lignes horizontales pour dessiner. On peut alors effacer texte et graphique séparément à l'aide des deux procédures suivantes :

Effacement du graphique

```
100 DEFPROC CLG
110 A=PEEK(25176)-10
120 FOR I=0 TO A STEP 5
130 PRINT@ 0,I;
135 VDU 30,31
140 NEXT I
150 VDU 23
160 ENDPROC
```

Effacer du texte

```
100 DEFPROC CLT
110 VDU 23
120 A=INT((PEEK(25177)-PEEK(25176))/10)
130 FOR I=1 TO A
140 VDU 30,31
150 NEXT I
160 ENDPROC
```

REMARQUES:

— On doit bien entendu éviter de dessiner dans la zone-texte pour séparer efficacement les deux fonctions.

— On peut définir une hauteur de WINDOW différente de celle donnée en exemple, car les procédures "CLT" et "CLG" se règlent automatiquement sur cette hauteur, en lisant dans les adresses contenant la taille de la fenêtre (voir § 3.1.2).

3. Il est également possible de protéger la ou les couleurs employées pour le graphique, puis y superposer du texte dans une couleur différente. Dans ce cas, CLS effacera le texte séparément du graphique.

Exemple :

```
100 REM TEXTGRAPH
102 PROTECT 0
105 CLS
110 INK 2
140 FOR I=0 TO 240 STEP 20
145 MOVE 0,0
150 PLOT 2,240,I
155 MOVE 0,240
156 PLOT 2,240,I
160 NEXT I
170 FOR I=0 TO 240 STEP 20
180 MOVE 0,0
190 PLOT 2,I,240
200 MOVE 0,240
210 PLOT 2,I,0
220 NEXT I
230 PROTECT 2
240 INK 7
250 PRINT"LE TEXTE S'ECRIT SUR LE GRAPHIQUE SANS EFFACER CE
DERNIER :"
255 VDU 24
256 PRINT
```

```
260 FOR I=1 TO 120
270 PRINT"Z";
280 NEXT I
300 VDU 25
310 PRINT@ 3,245;
```

Explications:

1ØØ,11Ø: Effface l'écran et sélectionne la couleur d'écriture.

14Ø,22Ø: Affichage du graphique.

23Ø : Protection de sa couleur.

24Ø,31Ø: Écriture de texte dans une autre couleur (superposition). Si un CLS est entré après exécution du programme, seul le texte est effacé.

4.1.5. Graphique et langage-machine

Dessiner par l'intermédiaire des accès directs à la mémoire présente l'avantage de la rapidité, avantage apprécié sur un ordinateur comme le Lynx dont l'affichage n'est pas des plus véloces...

Deux méthodes sont présentées, la saisie clavier par les ports du Z 8Ø et le tracé de lignes par l'appel d'une routine de la ROM.

La saisie clavier

Certains ports du Z 8Ø permettent une lecture fugitive de l'état du clavier par l'instruction INP. Cette lecture est équivalente au KEYN du Basic, mais se révèle plus performante car plus rapide et détectant l'autorépétition des touches.

On peut dresser une liste des valeurs lues sur les ports selon la touches appuyée:

Si K=INP(1152), on a:

K=247 Touche <ESPACE> appuyée.
K=254 Touche <6>
K=255 Aucune action, ou touche sans effet.

Si K=INP(2432) on a :

K=251 "Flèche Gauche"
K=223 "Flèche Droite"
K=219 Les deux Flèches ci-dessus, actionnées simultanément.
K=254 <DELETE>
K=247 <RETURN>
K=253 Touche "]"
K=255 Aucune action, ou touche sans effet.

Si K=INP(128), on a :

K=239 "Flèche vers le haut"
K=223 "Flèche vers le bas"
K=2Ø7 Les deux touches ci-dessus actionnées simultanément.
K=127 Détecte une pression sur <SHIFT>
K=247 Détecte une pression sur <SHIFT LOCK>
K=111 Pression simultanée sur <SHIFT> et ↑
K=95 Pression simultanée sur <SHIFT> et ↓
K=231 Pression simultanée sur <SHIFT LOCK> et ↑
K=215 Pression simultanée sur <SHIFT LOCK> et ↓
K=254 Touche <1>
K=255 Pas d'action, ou touche autre touche.

REMARQUE : On peut lire plusieurs ports successivement, pour combiner toutes les possibilités de chacun d'entre eux.

Exemple :

Voici une adaptation du programme "Dessin clavier" (§ 4.1.2), avec saisie des quatre flèches pour dessiner. L'emploi des ports 2432 et 128 simultanément permet le dessin en diagonale quand deux touches sont appuyées en même temps. La vitesse de tracé est étonnante !

```
100 REM DESSIN CLAVIER 2
110 TEXT
120 WINDOW 3,123,205,245
130 VDU 23
140 PRINT"UTILISEZ LES 4 FLECHES POUR DESSINER"
150 PRINT"AVEC 2 FLECHES, ON TRACE EN DIAGONALE"
160 PRINT"<SPACE> FIN DE SAISIE"
170 X=127,Y=102
190 A=INP(2432)
200 X=(X+(A=223))-(A=251)
```

```
210 A=INP(128)
220 Y=((Y+(A=223))-(A=239)) MOD 198
230 A=INP(1152)
240 IF A=247 THEN END
250 DOT X,Y
260 GOTO 190
```

Explications:

110,180: Initialisations, texte de présentation.

190 : Lecture du port 2432.

200 : Détection de l'appui sur "Flèche gauche" ou "Flèche droite". X est modifié en fonction du sens de déplacement horizontal choisi.

Si → est appuyée, l'expression A=223 est vraie, et vaut 1.

Si ← est appuyée, l'expression A=251 est vraie et vaut 1

Si aucune touche n'est utilisée, les deux expressions valent Ø.

210 : Lecture du port 128.

220 : Comme précédemment, détection de l'appui sur "Flèche haut" ou "Flèche bas". Y est modifié selon le même principe, en fonction du sens de déplacement choisi. La division MOD 198 empêche le débordement sur la partie texte du bas de l'écran.

230,240: Lecture du port 1152, et arrêt du programme si la touche <ESPACE> est appuyée.

250 : Affichage d'un point aux nouvelles coordonnées.

260 : Retour au début de la boucle.

Le tracé de lignes

Une routine de la ROM est utilisable en Basic pour tracer des droites: CALL 25186. Lors de l'appel de cette routine, une droite est tracée entre le point dont les coordonnées X1 et Y1 sont placées aux adresses 25189 et 25191, et le point dont les coordonnées X2 et Y2 sont placées aux adresses 25192 et 25194.

Une fois cette droite tracée, les coordonnées X2 et Y2 sont transférées dans les adresses 25189 et 25191, à la place de X1 et Y1, et donc mémorisées comme point d'origine pour le tracé suivant.

Exemple :

Tracé de deux droites :

```
10 CLS
20 POKE 25189,0
30 POKE 25191,0
40 POKE 25192,100
50 POKE 25194,100
60 CALL 25186
70 POKE 25192,200
80 POKE 25194,50
90 CALL 25186
```

Explication : On trace d'abord la droite allant de Ø,Ø à 1ØØ,1ØØ, puis de 1ØØ,1ØØ à 2ØØ,5Ø.

On peut adapter le programme "DEMO DRAW" (§ 4.1.1), pour afficher les points selon cette méthode :

```
100 REM DEMO DRAW 2
102 TEXT
110 FOR I=0 TO 2*PI STEP 0.1
120 POKE 25189,100
125 POKE 25191,100
130 y=100+(SIN(I)*50)
140 x=100+(COS(I)*50).
150 POKE 25192,x
160 POKE 25194,y
170 CALL 25186
180 NEXT I
```

Explication : Les lignes de 15Ø à 17Ø incluses remplacent l'instruction DRAW de "DEMO DRAW", et les lignes 12Ø et 125 l'instruction MOVE.

4.2. Le son du Lynx

Votre ordinateur peut très simplement émettre des notes de musique et des sons, grâce aux instructions BEEP et SOUND.

4.2.1. BEEP

Syntaxe : BEEP <période>,<durée>,<volume>

Application : Émet un son défini par trois paramètres.

— La période correspond à la hauteur du son, et doit être comprise entre Ø et 65535. Celle-ci est l'inverse de la fréquence du son, donc plus elle est élevée, plus le son est grave

— La durée peut être réglée entre Ø et 65535, il s'agit du nombre de périodes.

— Le volume peut être réglé entre Ø et 63.

REMARQUE :

— La valeur de <période>*<durée> doit être constante pour obtenir des sons d'égale longueur pour des hauteurs différentes.

Exemple :

```
100 REM GAMME
110 R=8,L=30000
130 WHILE R>=1/8
140 FOR I=1 TO 12
150 READ H
160 BEEP H*R,L/(H*R),63
170 NEXT I
180 R=R/2
190 RESTORE
200 WEND
210 END
220 DATA 415,393,369,351,332,311,293,276.5,262,249,234,221
```

Explications :

11Ø : Initialisations.

13Ø : Boucle d'émission des notes.

14Ø : Boucle de lecture des DATA (les douze notes de la gamme).

16Ø : Émission d'une note de longueur constante.

18Ø : Monter d'un octave.

19Ø : Réinitialise les DATA.

On peut dresser un tableau des notes de la gamme, pour l'octave habituellement utilisée (LA=44Ø hertz), avec N=Note et T=Période :

N	DO	DO#	RE	RE#	MI	FA	FA#	SOL	SOL#	LA	LA#	SI	DO
T	415	393	369	351	332	311	293	276.5	262	249	234	221	2Ø7.5

On descend et on monte d'un Octave en multipliant ou en divisant ces valeurs par 2.

Les effets sonores sont possible en incluant BEEP à l'intérieur de boucles :

```
100 REM MONTEE
110 FOR J=100 TO 10 STEP -1
120 BEEP J,10,63
130 NEXT J

100 REM PARTIE GAGNEE
110 FOR I=1 TO 20
120 BEEP RAND(400)+200,RAND(10)+20,63
130 NEXT I

100 REM R2D2
110 FOR J=0 TO 10
120 BEEP RAND(50)+50,20,63
130 NEXT J
```

4.2.2. SOUND

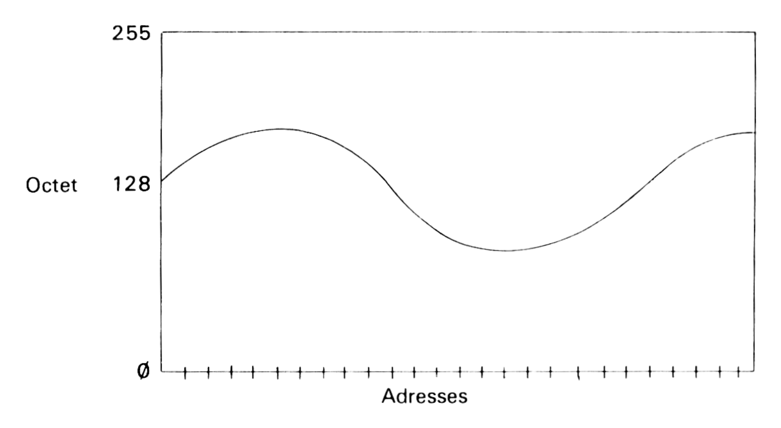
L'instruction SOUND est plus complexe, son emploi nécessitant une réservation d'espace-mémoire, et l'écriture d'une suite de codes correspondant à l'enveloppe du son que l'on désire émettre.

Syntaxe : SOUND <adresse>,<délai>

Application : SOUND lit successivement les octets à partir de <adresse>, et les convertit en son. <Délai> permet de fixer le temps entre la lecture de deux adresses. L'exécution de SOUND est interrompue lorsqu'une adresse contient la valeur Ø.

REMARQUE : En fait, SOUND autorise la création d'une forme d'onde dont on peut spécifier tous les paramètres. On utilise ensuite <délai> pour régler la hauteur du son (ou du bruit !) émis.

Une forme d'onde peut se définir par un dessin, qui représente son enveloppe. Soit par exemple une sinusoïde simple : La valeur la plus basse qu'un octet peut prendre est égale à 1, la plus haute à 255. Si l'on dessine une grille correspondant à ces limites, avec les adresses en abscisses et leur contenu en ordonnées, on peut y tracer la courbe et en déduire la valeur de tous les octets la composant :



Une fois constituée la liste des octets correspondant à la courbe, il faut les placer successivement dans une zone-mémoire réservée à l'aide de POKE :

```
100 RESERVE HIMEM-22
110 FOR I=0 TO 21
120 READ R
130 POKE HIMEM+I,R
140 NEXT I
150 END
160 DATA 140,160,162,164,162,160,156,150,135,125,120,110,100,90,
95,102,112,130,140,0
```

Explication :

1ØØ : On décale le HIMEM de 22 octets.

11Ø : Boucle de lecture des DATA (forme de la courbe).

12Ø : Inscription des valeurs lues dans DATA à partir de HIMEM.

L'émission des sons se fait ensuite par SOUND HIMEM,<délai>. La courbe définie dans ce programme ne comportant qu'une seule alternance, il faut l'inclure dans une boucle pour obtenir un son continu :

```
1000 FOR I=1 TO 1000
1010 SOUND HIMEM,10
1020 NEXT I
```

En modifiant le délai, on peut changer la hauteur du son :

```
1000 FOR I=1 TO 100
1010 SOUND HIMEM,I
1020 NEXT I
```

REMARQUE :

— La forme d'onde présentée est très simple, mais rien n'empêche le dessin d'une enveloppe complexe sur un plus grand nombre d'octets, la seule limite étant la capacité mémoire du Lynx.

— On peut employer SOUND pour lire des adresses n'importe où dans la mémoire, le résultat est parfois surprenant. En voici quelques exemples :

```
100 REM MITRAILLETTE
110 FOR I=1 TO 10
120 SOUND 1850,70
130 PAUSE 50
140 NEXT I

100 REM HELICO
110 FOR I=500 TO 10 STEP -10
120 SOUND 2200,I
130 PAUSE RAND(100)
140 NEXT I

100 REM VAISSEAU SPATIAL
110 FOR I=1 TO 50
120 SOUND 1850,I
140 NEXT I

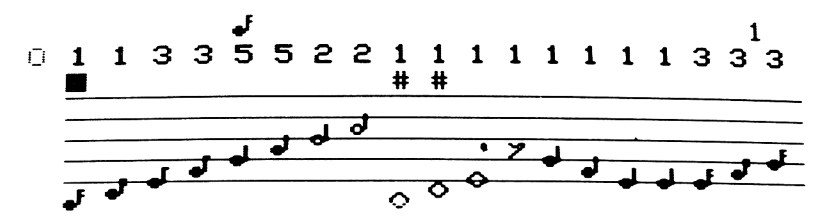
100 REM DESTRUCTION
110 FOR I=500 TO 100 STEP -100
120 SOUND 1850,I
130 PAUSE RAND(100)
140 NEXT I

100 REM MOTO
110 FOR I=1850 TO 2100
120 SOUND I,1
140 NEXT I

100 REM ZIP
110 FOR I=1 TO 100
120 SOUND 2400,I
140 NEXT I
```

4.2.3. Le programme "CHOPIN"

PROGRAMME CHOPIN BURP




```
←→       :  LONGUEUR NOTE (♪♪♩♩𝅗𝅥○)
↑↓       :  SELECTION OCTAVE (1-5)
1-8      :  SELECTION NOTE (#: SHIFT)
ESPACE   :  SILENCE (𝄾)
< >      :  DEFILEMENT PARTITION
I        :  INSERER UNE NOTE
DEL      :  EFFACER LA NOTE
@        :  JOUER LA PAGE AFFICHEE
J        :  JOUER LA MELODIE EN MEMOIRE
E        :  EFFACER LA MELODIE
P        :  IMPRIMER LA MELODIE
S ET C   :  SAUVEGARDE ET CHARGEMENT K7
```

Ce programme de création musicale exploite au maximum les possibilités du Basic, dans la mesure où il regroupe des instructions de couleur, du graphique, ainsi que les routines de création de caractères. Son fonctionnement sur la version 48K du Lynx implique quelques modifications (voir remarques).

Il vous permettra :

1. De saisir et mémoriser une mélodie à partir du clavier. Les touches de 1 à 8 sont affectées aux notes de la gamme, la touche <ESPACE> au silence et <SHIFT> permet l'obtention des dièses. La mélodie est affichée au fur et à mesure à l'écran sur une portée musicale. La longueur et la hauteur des notes peuvent être constamment modifiées.

2. De jouer la mélodie mémorisée au tempo choisi.

3. De modifier la mélodie directement sur la portée à l'aide de commandes simples. Un curseur pointe constamment la note en cours.

4. De copier la mélodie en mémoire sur une imprimante.

5. La sauvegarde et le chargement d'une mélodie sur K7.

Commandes disponibles :

← → Modifie la longueur de la note et des silences. Cinq valeurs sont disponibles, et correspondent à double croche, croche, noire, blanche et ronde. La longueur en cours est constamment affichée à l'écran. Toute note modifiée et toute nouvelle entrée se fait en fonction de cette longueur.

↓ ↑ Permet de descendre ou de monter de un ou plusieurs Octaves. L'Octave en cours est affiché à l'écran, au dessus de chaque note de la partition.

I Après action sur I, un silence est inséré à l'intérieur de la mélodie et de la partition, à la position du curseur. Pour insérer une note, il suffit de surcharger ce silence avec la note désirée.

<DEL> Supprime de la mélodie la note présente sous le curseur.

< > Autorise le déplacement à l'intérieur de la partition. Lorsque le curseur arrive en bout d'écran, le curseur se place sur la note suivante de la page suivante. Le numéro de la page en cours est constamment affiché à l'écran. Les notes sur lesquelles passe le curseur peuvent être directement modifiées, sans autres commandes.

@ Joue à un tempo fixe les notes de la page affichée à l'écran (permet le contrôle immédiat des modifications).

E Efface la mélodie en mémoire, pour permettre une nouvelle entrée.

J Rejoue la mélodie mémorisée, selon le tempo désiré.

P Copie les notes composant la mélodie sur l'imprimante.

S,C Sauvegarde et chargement sur K7.

REMARQUES :

— Le nombre de notes composant la mélodie a été limité à 5ØØ sur la version 96K, le programme interdit toute nouvelle entrée une fois cette limite atteinte.

— On peut, d'une note à l'autre, sauter de plusieurs Octaves à la fois.

— Toute commande effaçant la mélodie doit être confirmée.

— Le programme est prévu pour une imprimante branchée sur l'interface parallèle. Dans le cas d'un interface série, il faut remplacer la ligne 333Ø par :

333Ø EXT SPRINT

Sur le Lynx 48K., il faut charger le programme SPRINT pour rendre cette instruction fonctionnelle. Il est à noter que la commande P suspend l'exécution du programme si aucune imprimante n'est connectée.

— La sauvegarde sur cassette utilise le Moniteur, avec sortie du Basic : Attention donc aux fausses manœuvres !

— Pour le Lynx 48K., le nombre de notes que l'on peut entrer est malheureusement limité à 5Ø, le programme occupant à lui seul 12K. sur les 13.5K. réellement disponibles. Pour utiliser cette version, il faut écrire :

```
210 DIM M(51*6)

1110 q=40651

1120 RESERVE 40580

1350 IF X>48 AND X<57 AND P<50 OR X>31 AND X<40 AND P<50
     THEN GOSUB LABEL SAISIE

3280 IF S<50 THEN LET S=S+1

3830 PRINT "TAPEZ: D 9E84 9FF6 0 'MELODIE'"
```

```
100 REM CHOPIN
110 GOSUB  LABEL INITIALISATION
120 GOSUB  LABEL VARIABLES
130 GOSUB  LABEL ECRAN
140 GOSUB  LABEL EFFACER CADRE
150 GOSUB  LABEL INSCRIPTIONS
160 GOSUB  LABEL VARIABLES 2
170 GOTO  LABEL BOUCLE
180 END
190 LABEL VARIABLES
200 LET A=0,B=1,C=2,D=3
210 DIM M(501*6)
220 LET O=3,L=130,T=13250
230 LET P=1,H=33,V=100,h=9,v=100,S=1
240 LET l=35,o=45,c=55
250 LET z=0,m=3,y=0,n=0,u=1
260 LET X=0,F=0
270 DIM D$(4)(7)
```

```
280 FOR I=1 TO 7
290   READ D$(I)
300 NEXT I
310 DIM N$(4)(8)
320 FOR I=1 TO 8
330   READ N$(I)
340 NEXT I
350 DIM N(9*3)
360 FOR J=1 TO 2
370   FOR I=1 TO 8
380     READ N(I*3+J)
390   NEXT I
400 NEXT J
410 DIM O(5),L(5)
420 FOR I=1 TO 5
430   READ O(I),L(I)
440 NEXT I
450 DIM L$(15)(5)
460 FOR I=1 TO 5
470   READ L$(I)
480 NEXT I
490 RETURN
500 LABEL VARIABLES 2
510 PRINT  @ H,l;CHR$(L);
520 PRINT  @ 114,l;u;" ";
530 PRINT  @ h,o;O;
540 PRINT  @ h,c;CHR$(134);
550 RETURN
560 LABEL ECRAN
570 PROTECT 0
580 PAPER A
590 WINDOW 3,123,5,245
600 CLS
610 PAPER B
620 PRINT  @ 3,5
630 PRINT  @ 3,35
640 FOR I=55 TO 85 STEP 10
650   PRINT  @ 9,I
660 NEXT I
670 PAPER A
680 INK D
690 FOR J=65 TO 97 STEP 8
700   FOR I=18 TO 245
710     DOT I,J
720   NEXT I
730 NEXT J
740 PROTECT B
750 RETURN
760 LABEL EFFACER CADRE
770 PROTECT 0
780 PAPER B
790 WINDOW 3,123,125,245
800 VDU 23
```

```
810 FOR I=1 TO 12
820   VDU 30,31
830 NEXT I
840 PROTECT B
850 RETURN
860 LABEL INSCRIPTIONS
870 INK D
880 PRINT  @ 30,15;"PROGRAMME CHOPIN BVRP";
890 PRINT  @ 3,35;"LONGUEUR:";
900 PRINT  @ 78,35;"PAGE NUMERO ";
910 PRINT  @ 3,45;"O";
920 PRINT  @ 3,55;">";
930 VDU 23
940 PRINT CHR$(124);CHR$(123);"       : LONGUEUR NOTE (";CHR$(128);CHR$(129);CHR$
(130);CHR$(131);CHR$(132);")       "
950 PRINT CHR$(125);CHR$(126);"       : SELECTION OCTAVE (1-5)"
960 PRINT "1-8      : SELECTION NOTE (#:SHIFT)"
970 PRINT "ESPACE   : SILENCE (";CHR$(133);")"
980 PRINT "< >      : DEFILEMENT PARTITION "
990 PRINT "I        : INSERER UNE NOTE"
1000 PRINT "DEL      : EFFACER LA NOTE"
1010 PRINT "@        : JOUER LA PAGE AFFICHEE"
1020 PRINT "J        : JOUER LA MELODIE EN MEMOIRE"
1030 PRINT "E        : EFFACER LA MELODIE"
1040 PRINT "P        : IMPRIMER LA MELODIE"
1050 PRINT "S ET C   : SAUVEGARDE ET CHARGEMENT K7";
1060 PROTECT D
1070 INK 7
1080 RETURN
1090 LABEL INITIALISATION
1100 DPOKE 25139,1024
1110 LET q=62511
1120 RESERVE 62440
1130 DPOKE GRAPHIC,HIMEM
1140 FOR J=0 TO 69
1150   READ R
1160   POKE LETTER(128)+J,BIN(R)
1170 NEXT J
1180 CCHAR 256*32+134
1190 RETURN
1200 DATA 000011,000010,000011,000010,011010,111110,111100,011000,000000,000000
1210 DATA 000011,000010,000010,000010,011010,111110,111100,011000,000000,000000
1220 DATA 000010,000010,000010,000010,011010,111110,111100,011000,000000,000000
1230 DATA 000010,000010,000010,000010,011010,100110,100100,011000,000000,000000
1240 DATA 000000,000000,001100,010010,100001,100001,010010,001100,000000,000000
1250 DATA 00000,000000,010011,001101,000010,000100,001000,010000,000000,000000
1260 DATA 000000,111111,111111,111111,111111,111111,111111,111111,000000,000000
1270 DATA DO#,RE#,FA,FA#,SOL#,LA#,DO
1280 DATA DO,RE,MI,FA,SOL,LA,SI,DO
1290 DATA 415,369,332,311,276.5,249,221,207.5
1300 DATA 393,351,311,293,262,234,207.5,196.5
1310 DATA 4,0.25,2,0.5,1,1,0.5,2,0.25,4
1320 DATA DOUBLE CROCHE,CROCHE,NOIRE,BLANCHE,RONDE
```

```
1330 LABEL BOUCLE
1340 LET X=GETN
1350 IF X>48 AND X<57 AND P<500 OR X>31 AND X<40 AND P<500 THEN  GOSUB  LABEL SA
ISIE
1360 IF X=10 OR X=11 THEN  GOSUB  LABEL OCTAVE
1370 IF X=22 OR X=12 THEN  GOSUB  LABEL LONGUEUR
1380 IF X=44 OR X=46 THEN  GOSUB  LABEL DEFILEMENT
1390 IF X=69 OR X=101 THEN  GOSUB  LABEL ERASE
1400 IF X=74 OR X=106 THEN  GOSUB  LABEL REJOUER
1410 IF X=8 THEN  GOSUB  LABEL DELETE
1420 IF X=73 OR X=105 THEN  GOSUB  LABEL INSERT
1430 IF X=64 THEN  GOSUB  LABEL PLAYPAGE
1440 IF X=80 OR X=112 THEN  GOSUB  LABEL PRINTER
1450 IF X=83 OR X=115 THEN  GOSUB  LABEL SAVE
1460 IF X=67 OR X=99 THEN  GOSUB  LABEL LOAD
1470 GOTO  LABEL BOUCLE
1480 LABEL OCTAVE
1490 IF X=10 AND O<>1 THEN  LET O=O-1
1500 IF X=11 AND O<>5 THEN  LET O=O+1
1510 LET M(P*6+3)=O
1520 PRINT  @ h,o;O;
1530 RETURN
1540 LABEL LONGUEUR
1550 IF X=12 AND L<>132 THEN  LET L=L+1
1560 IF X=22 AND L<>128 THEN  LET L=L-1
1570 PRINT  @ H,l;CHR$(L);
1580 RETURN
1590 LABEL SAISIE
1600 IF X=32 THEN  GOTO 1650
1610 IF X>48 AND X<57 THEN  LET v=100-(4*(X-49)),M(P*6+1)=N((X-48)*3+1),M(P*6+5)
=X
1620 ELSE  LET v=100-(4*(X-33)),F=1,M(P*6+1)=N((X-32)*3+2),M(P*6+5)=X
1630 IF F=1 THEN  GOSUB  LABEL DIESE
1640 ELSE  PRINT  @ h,c;" ";
1650 IF X=32 THEN  LET X=X+101,v=81,M(P*6+1)=0,M(P*6+5)=X
1660 ELSE  LET X=L
1670 FOR I=70 TO 100 STEP 10
1680   PRINT  @ h,I;" ";
1690 NEXT I
1700 PRINT  @ h,v;CHR$(X);
1710 IF X=133 THEN  PRINT  @ h,c;" ";
1720 LET h=h+6
1730 LET M(P*6+2)=L,M(P*6+3)=O,M(P*6+4)=F
1740 GOSUB  LABEL EMISSION
1750 LET F=0,P=P+1
1760 IF S=P-1 THEN  LET S=S+1
1770 IF h<>123 THEN  GOTO 1800
1780 GOSUB  LABEL NEXTPAGE
1790 RETURN
1800 GOSUB  LABEL CURSEUR DIESE
1810 RETURN
1820 LABEL DIESE
1830 IF RIGHT$(D$(X-32),1)="#" THEN  GOTO 1870
```

```
1840 PRINT  @ h,c;" ";
1850 LET v=v-4
1860 RETURN
1870 PRINT  @ h,c;"#";
1880 RETURN
1890 LABEL EMISSION
1900 IF M(P*6+1)<>0 THEN  BEEP M(P*6+1)*O(M(P*6+3)),T*L(M(P*6+2)-127)/(M(P*6+1)*
O(M(P*6+3))),63
1910 ELSE  BEEP 1000,T*L(M(P*6+2)-127)/1000,0
1920 RETURN
1930 LABEL PAGE
1940 CLS
1950 LET h=9
1960 FOR I=P-m TO P-n
1970   PRINT  @ h,o;M(I*6+3);
1980   LET X=M(I*6+5)
1990   IF X=133 THEN  GOTO 2110
2000   LET F=M(I*6+4)
2010   IF F=0 THEN  LET v=100-(4*(X-49))
2020   ELSE  LET v=100-(4*(X-33))
2030   IF F=1 THEN  GOSUB  LABEL DIESE
2040   LET X=M(I*6+2)
2050   PRINT  @ h,v;CHR$(X);
2060   LET p=P,P=I
2070   GOSUB  LABEL EMISSION
2080   LET h=h+6,P=p
2090 NEXT I
2100 RETURN
2110 LET v=81
2120 GOTO 2050
2130 LABEL DEFILEMENT
2140 LET x=X
2150 IF P<>S THEN  GOTO 2180
2160 PRINT  @ h,c;" ";
2170 GOTO 2210
2180 LET F=M(P*6+4),X=M(P*6+5)
2190 IF F=1 THEN  GOSUB  LABEL DIESE
2200 ELSE  PRINT  @ h,c;" ";
2210 LET y=0,z=0
2220 IF x=44 AND h=9 AND P<>1 THEN  GOSUB  LABEL PRECEDPAGE
2230 IF y=1 THEN  GOTO 2250
2240 IF x=44 AND h<>9 AND P<>1 THEN  LET h=h-6,P=P-1
2250 IF x=46 AND h=117 AND P<>S THEN  GOSUB  LABEL NEXTPAGE
2260 IF z=1 THEN  GOTO 2280
2270 IF x=46 AND h<>117 AND P<>S THEN  LET h=h+6,P=P+1
2280 GOSUB  LABEL CURSEUR DIESE
2290 RETURN
2300 LABEL CURSEUR DIESE
2310 IF P=S THEN  GOTO 2400
2320 LET O=M(P*6+3),F=M(P*6+4)
2330 IF F<>1 THEN  GOTO 2400
2340 IF NOTRIGHT$(D$(M(P*6+5)-32),1)="#" THEN  GOTO 2400
2350 VDU 18
```

```
2360 PRINT  @ h,c;"#";
2370 VDU 18
2380 LET F=0
2390 RETURN
2400 LET F=0
2410 GOSUB  LABEL VARIABLES 2
2420 RETURN
2430 LABEL PRECEDPAGE
2440 LET P=P+2,m=18
2450 GOSUB  LABEL PAGE
2460 LET h=99,y=1,P=P-3,u=u-1
2470 LET F=M(P*6+4),O=M(P*6+3)
2480 GOSUB  LABEL VARIABLES 2
2490 RETURN
2500 LABEL NEXTPAGE
2510 IF S>P THEN  GOTO 2530
2520 LET P=P-1
2530 IF P+16<S THEN  LET n=0,P=P+16,m=18
2540 ELSE  LET m=S-P+2,P=S,n=1
2550 GOSUB  LABEL PAGE
2560 IF P=S THEN  PRINT  @ h,o;O;
2570 LET h=27,z=1,P=P-m+3,n=0,u=u+1
2580 IF P<>S THEN  LET F=M(P*6+4),O=M(P*6+3)
2590 GOSUB  LABEL VARIABLES 2
2600 RETURN
2610 LABEL ERASE
2620 IF S<>1 THEN  GOTO  LABEL ATTENTION
2630 LET P=1,S=1,u=1,h=9,O=3,L=130
2640 CLS
2650 GOSUB  LABEL VARIABLES 2
2660 RETURN
2670 LABEL REJOUER
2680 IF S=1 THEN  RETURN
2690 PRINT  @ H,l;" ";
2700 PRINT  @ h,c;" ";
2710 GOSUB  LABEL EFFACER CADRE
2720 INK D
2730 VDU 23
2740 PRINT "ENTREZ LE TEMPO DESIRE, PUIS <RETURN>"
2750 PRINT
2760 PRINT "( EN BATTEMENTS/MINUTE DE LA NOIRE )"
2770 PRINT
2780 INPUT "TEMPO (10-1000) ";t
2790 IF t<10 OR t>1000 THEN  GOTO 2750
2800 LET T=20000*318/t
2810 PROTECT D
2820 INK 7
2830 LET p=P
2840 FOR P=1 TO S-1
2850   GOSUB  LABEL EMISSION
2860 NEXT P
2870 PROTECT 0
2880 PROTECT B
```

```
2890 GOSUB  LABEL INSCRIPTIONS
2900 LET P=p,T=13250
2910 GOSUB  LABEL VARIABLES 2
2920 GOSUB  LABEL CURSEUR DIESE
2930 RETURN
2940 LABEL DELETE
2950 IF S=1 OR P=S THEN  RETURN
2960 IF S<>2 THEN  GOTO 2990
2970 GOSUB  LABEL ERASE
2980 RETURN
2990 FOR I=P TO S-1
3000   LET J=1
3010   WHILE I+1<>S AND J<>6
3020     LET M(I*6+J)=M((I+1)*6+J)
3030     LET J=J+1
3040   WEND
3050 NEXT I
3060 LET S=S-1,F=0
3070 IF (S-19) MOD 16=0 AND u<>1 THEN  LET u=u-1,h=99+(((P-16) MOD 16)*6)
3080 GOSUB  LABEL PLAYPAGE
3090 RETURN
3100 LABEL PLAYPAGE
3110 IF S=1 THEN  RETURN
3120 LET f=h,g=P
3130 LET P=(u*16)+3,m=18
3140 IF P>=S THEN  LET P=S-1,m=P-((u-1)*16+1)
3150 GOSUB  LABEL PAGE
3160 IF P=S-1 AND h<>123 THEN  PRINT  @ h,o;O;
3170 LET h=f,P=g
3180 GOSUB  LABEL VARIABLES 2
3190 GOSUB  LABEL CURSEUR DIESE
3200 RETURN
3210 LABEL INSERT
3220 IF S=P THEN  RETURN
3230 FOR I=S TO P+1 STEP -1
3240   FOR J=1 TO 5
3250     LET M(I*6+J)=M((I-1)*6+J)
3260   NEXT J
3270 NEXT I
3280 IF S<500 THEN  LET S=S+1
3290 LET M(P*6+1)=0,M(P*6+4)=0,M(P*6+5)=133
3300 GOSUB  LABEL PLAYPAGE
3310 RETURN
3320 LABEL PRINTER
3330 EXT PPRINT
3340 LPRINT
3350 LPRINT
3360 LPRINT "COPIE DE LA MELODIE :"
3370 LPRINT
3380 IF S=1 THEN  RETURN
3390 FOR I=1 TO S-1
3400   IF M(I*6+1)<>0 THEN  GOTO 3430
3410   LET A$="SILENCE"
```

```
3420   GOTO 3470
3430   IF M(I*6+4)<>0 THEN  GOTO 3460
3440   LET A$=N$(M(I*6+5)-48)
3450   GOTO 3470
3460   LET A$=D$(M(I*6+5)-32)
3470   LPRINT
3480   LPRINT "NOM DE LA NOTE ";I;" : ";A$;"      LONGUEUR : ";L$(M(I*6+2)-127);
3490   IF NOTA$="SILENCE" THEN  LPRINT "      OCTAVE : ";M(I*6+3)
3500   ELSE  LPRINT
3510   LET p=P,P=I
3520   GOSUB  LABEL EMISSION
3530   LET P=p
3540 NEXT I
3550 LPRINT
3560 LPRINT
3570 RETURN
3580 LABEL ATTENTION
3590 PRINT  @ 3,110;"VOUS CONFIRMEZ LA COMMANDE (O/N)?";
3600 LET k=GETN
3610 IF k=78 OR k=110 THEN  GOTO 3640
3620 IF k=79 OR k=111 THEN  GOTO 3670
3630 GOTO 3600
3640 PRINT  @ 3,110;CHR$(30);CHR$(31);
3650 LET n=0
3660 RETURN
3670 IF n=0 THEN  GOTO 2630
3680 ELSE  GOTO 4010
3690 LABEL SAVE
3700 IF S=1 THEN  RETURN
3710 GOSUB  LABEL EFFACER CADRE
3720 LET Z=q
3730 DPOKE Z,S
3740 LET Z=Z+2
3750 FOR I=1 TO S-1
3760   DPOKE Z,(M(I*6+1)*10)
3770   FOR J=2 TO 5
3780     POKE Z+J,M(I*6+J)
3790   NEXT J
3800   LET Z=Z+6
3810 NEXT I     .
3820 VDU 23
3830 PRINT "TAPEZ:D F3E8 FFF8 0 'MELODIE'"
3840 PRINT
3850 PRINT "METTEZ LE K7 EN RECORD"
3860 PRINT "PUIS VALIDEZ PAR <RETURN>"
3870 PRINT
3880 PRINT "POUR RETOURNER AU PROGRAMME, TAPEZ: "
3890 PRINT "J PUIS <RETURN> SUIVI DE GOTO 3940"
3900 INK D
3910 WINDOW 3,123,235,245
3920 VDU 23
3930 MON
3940 WINDOW 3,123,125,245
```

```
3950 INK 7
3960 GOSUB  LABEL INSCRIPTIONS
3970 RETURN
3980 LABEL LOAD
3990 LET n=1
4000 GOTO  LABEL ATTENTION
4010 LET n=0
4020 CLS
4030 GOSUB  LABEL EFFACER CADRE
4040 VDU 23
4050 PRINT "POSITIONNEZ LA K7"
4060 PRINT
4070 PRINT "METTEZ SUR PLAY..."
4080 PRINT
4090 MLOAD "MELODIE"
4100 LET Z=q
4110 LET S=DPEEK(Z)
4120 LET Z=Z+2
4130 FOR I=1 TO S-1
4140   LET M(I*6+1)=DPEEK(Z)/10
4150   FOR J=2 TO 5
4160     LET M(I*6+J)=PEEK(Z+J)
4170   NEXT J
4180   LET Z=Z+6
4190 NEXT I
4200 GOSUB  LABEL INSCRIPTIONS
4210 CLS
4220 LET P=1,u=1,h=9,O=3,L=130,F=0
4230 GOSUB  LABEL PLAYPAGE
4240 RETURN
```

Structure et explications du programme :

Comme GENECAR, CHOPIN est constitué d'une succession de routines appelées en série par le programme source (lignes 1ØØ-18Ø) :

Nom du label	*Fonction*
INITIALISATION	Réservation de la zone-mémoire nécessaire à la création des notes de musique et à la sauvegarde de la mélodie sur K7. Création des caractères spéciaux par lecture de lignes de DATA.
VARIABLES	Initialisations des diverses variables et tableaux du programme.
ECRAN	Présentation écran de base.

EFFACER CADRE	Création et effacement d'une fenêtre-texte destinée à l'affichage des commandes disponibles.
INSCRIPTIONS	Affichage du titre et des commandes disponibles.
VARIABLES 2	Sous-programme d'affichage de la longueur de note et de l'octave en cours, du curseur et du numéro de la page.
LABEL BOUCLE	Boucle de saisie des commandes. Celle-ci appelle elle-même un certain nombre de sous-routines, en fonctions des commandes entrées.
Sous-routines:	
SAISIE	Saisie, affichage et émission d'une nouvelle note de musique.
OCTAVE	Modification de l'octave en cours.
LONGUEUR	Modification de la durée de note en cours.
DEFILEMENT	Sous-routine autorisant le déplacement à l'intérieur de la partition.
ERASE	Efface la mélodie en mémoire, après confirmation.
REJOUER	Pour écouter toute la mélodie au tempo désiré.
DELETE	Efface la note sous le curseur.
INSERT	Insertion d'un silence dans la partition à la position du curseur.
PLAYPAGE	Joue la page affichée à l'écran, à un tempo fixe.
PRINTER	Copie sur l'imprimante la mélodie en mémoire.
SAVE	Sauvegarde sur K7 de la mélodie, en recopiant celle-ci dans la zone-mémoire réservée (INITIALISATION), puis en chargeant cette zone à l'aide de la commande DUMP du Moniteur.
LOAD	Recharge la mélodie en mémoire à partir du K7.

Enfin, d'autres sous-programmes sont fréquemment appelés par toutes les routines du programme:

DIESE	Affiche un signe "#" à la position du curseur, si la note sélectionnée correspond à un dièse.
EMISSION	Émet la note définie par P, sa position dans la mélodie.
PAGE	Affiche à l'écran la page sélectionnée.

CURSEUR DIESE	Conserve l'affichage du signe "#", même lors des déplacements du curseur.
PRECEDPAGE	Affiche la page précédente, quand le curseur dépasse la première note de la page en cours en mode défilement.
NEXTPAGE	Passe à la page suivante (automatique en mode saisie).
ATTENTION	Message réclamant une confirmation de type O/N, pour les commandes effaçant une mélodie en mémoire.

REMARQUE: Toutes les routines de "CHOPIN" sont présentées ci-dessus, mais chacune d'elles en appelle d'autres en fonction des besoins du programme.

Appendice : Les messages d'erreur

Code : 1

Message : OUT OF MEMORY

Signification : La totalité de la mémoire vive disponible a été occupée, ou vous avez tenté de charger un programme trop long à partir d'une cassette.

Code : 2

Message : WRONG MODE

Signification : Tentative d'utiliser en mode direct des commandes ne fonctionnant que dans un programme (INPUT, etc.).

Code : 3

Message : STRING ERROR

Signification : Indique une erreur concernant une chaîne, comme par exemple la tentative d'y attribuer plus de 127 caractères.

Code : 4

Message : MISSING BRACKET

Signification : Un nombre impair de guillemets a été rencontré dans une ligne de programme.

Code : 5

Message : DIVIDE BY ZERO ERROR

Signification : Tentative de diviser un nombre par Ø.

Code : 6

Message : OVERFLOW ERROR

Signification : Un nombre supérieur à 9.9 $1Ø^{63}$ a été rencontré (ou calculé) dans un programme.

Code : 7

Message : SYNTAX ERROR

Signification : Une erreur de syntaxe a été rencontrée dans une commande directe ou une ligne de programme.

Code : 8

Message : SOMETHING MISSING

Signification : Il manque un opérateur ou une opérande dans une ligne de programme ou une commande directe.

Code : 9

Message : FUNCTION ARGUMENT ERROR

Signification : Un argument erroné ou hors limites a été fourni à une fonction, par ex. SQR(−1). (Racine carrée de −1).

Code : 11

Message : REDIMENSIONED ARRAY

Signification : Réponse à une tentative de redimensionner un tableau déjà déclaré par DIM.

Code: 12

Message: LINE, LABEL OR PROC NOT FOUND

Signification: Un numéro de ligne ou LABEL inexistant a été fourni à GOTO, THEN ou GOSUB, ou alors un PROC se réfère à une procédure inexistante.

Code: 13

Message: NUMBER OUT OF RANGE

Signification: Un nombre hors limites a été fourni, ou rencontré lors d'un calcul.

Code: 14

Message: SUBSCRIPT OUT OF RANGE

Signification: Un élément de tableau est appelé en utilisant un indice hors des limites déclarées dans DIM, lorsque le tableau a été dimensionné.

Code: 15

Message: TYPE MISMATCH

Signification: Discordance entre un type de variable et les données que l'on essaie de lui attribuer (par ex: A="ABCD"), ou entre une commande et l'argument qu'on lui donne (ex: SIN(A$)).

Code: 16

Message: LINE TOO LONG

Signification: Apparaît en mode direct, si la ligne de programme en cours d'écriture dépasse 24Ø caractères.

Code: 17

Message: CANNOT CONTINUE

Signification: Lorsqu'un programme a été arrêté par ESC, réponse à l'instruction CONT si une modification au programme est effectuée pendant l'interruption.

Code: 18

Message: OUT OF DATA

Signification : Tentative d'utiliser READ, alors que les lignes de DATA ont déjà été lues en entier.

Code : 19

Message : RETURN WITHOUT GOSUB

Signification : Un RETURN est rencontré sans être préalablement passé par un GOSUB, ou un GOTO a provoqué la sortie d'une boucle FOR-NEXT avant la fin de celle-ci.

Code : 2Ø

Message : NEXT WITHOUT FOR

Signification : Une commande NEXT est rencontrée sans que la boucle ait débuté par un FOR.

Code : 21

Message : UNDEFINED VARIABLE

Signification : Indique l'utilisation d'une variable, sans qu'une valeur lui ait été préalablement attribuée.

Code : 22

Message : UNTIL WITHOUT WITHOUT REPEAT

Signification : Un UNTIL est rencontré, sans que la boucle ait commencé par REPEAT.

Code : 23

Message : WEND WITHOUT WHILE

Signification : Un WEND est rencontré, sans que la boucle ait débuté par WHILE.

Code : 24

Message : WHILE WITHOUT WEND

Signification : Une boucle a été amorcée par WHILE, sans être terminée par un WEND.

Code: 25

Message: RETURN STACK FULL

Signification: Trop de boucles du type FOR-NEXT, REPEAT-UNTIL etc. imbriquées.

Code: 26

Message: REPEAT WITHOUT UNTIL

Signification: Une boucle débutant par REPEAT ne se termine pas par un UNTIL.

Code: 27

Message: GOSUB WITHOUT RETURN

Signification: Un sous-programme appelé par un GOSUB ne se termine pas par RETURN.

Code: 28

Message: ENDPROC WITHOUT PROC

Signification: Une commande ENDPROC est rencontrée, sans que le sous-programme ait été appelé par PROC.

Code: 29

Message: BAD TAPE

Signification: Une vérification du programme sauvegardé par VERIFY s'est avérée négative.

Index alphabétique

Achevé d'imprimer
par **Cid** *éditions*

Dépôt légal : avril 1984
N° d'Éditeur : 4096

Présenté sous la forme d'un manuel de Basic approfondi, ce livre est destiné à ceux qui, déjà au fait des rudiments du Basic, désirent aller plus loin dans la connaissance de leur ordinateur. De nombreux programmes, dont certains sont des logiciels complets et prêts à utiliser, permettent d'ouvrir la voie vers une programmation avancée.

Citons un générateur de caractères et un programme musical prévu pour mémoriser une mélodie, en éditer la partition et copier celle-ci sur imprimante ou cassette.

L'initiation au langage-machine tient une place importante, avec là encore des programmes pour permettre au lecteur, même novice, l'emploi de quelques « astuces » utiles sans connaissances de base. On trouvera, parmi les programmes proposés, une méthode de reconfiguration du clavier, ainsi que la routine à utiliser pour créer et exploiter deux pages-écran.



8213


TOUT SAVOIR SUR LYNX
B. VANRYB
R. POLITIS



Document numérisé avec amour par :

AMSTRAD CPC

MÉMOIRE ÉCRITE

https://acpc.me/